

042159.0119

COMPUTER PROGRAM LISTING APPENDIX

COMPUTER PROGRAM LISTING APPENDIX

COMPUTER PROGRAM LISTING APPENDIX

votc.h

```

/*
***** file: tc_v2.h ***** date: April 12, 2000
***** function: tests turbo codes ***** Modulation: QAM
***** Decoder: MAP algorithm *****
*/
#include <math.h>
#include <stdio.h>
#include <malloc.h>
#include <dos.h>

/* Definition of the first recursive systematic code (RSC1):
#define RSC1_ENC_MEM 4 /* encoder memory order */
#define RSC1_STATES (1 << RSC1_ENC_MEM)
#define RSC1_FP 035 /* forward polynomial in octal */
#define RSC1_BP 023 /* backward polynomial in octal */
*/
/* Definition of the second recursive systematic code (RSC2):
#define RSC2_ENC_MEM 4 /* encoder memory order */
#define RSC2_STATES (1 << RSC2_ENC_MEM)
#define RSC2_FP 035 /* forward polynomial in octal */
#define RSC2_BP 023 /* backward polynomial in octal */
*/
#define NR_ITER 8 /* nr. of iterative decoding stages */
#define EBNO 6.0 /* Eb/No in dB */
#define MAX_ERRORS 1000 /* stop when this nr. is reached */
#define INT_SIZE 6144 /* nr. of info bits to be interleaved */
#define MAX (exp(31.0)) /* limit soft outputs */
#define E_STEPS 1000 /* number of values for E_val */
#define PRINT_BLOCKS 100 /* how often to print results */
#define SEED1 13733 /* seeds for random nr. gen. */
#define SEED2 1935791

#define SIGMA_12_4AM sqrt(2.50 * pow(10.0,(-EBNO / 10.0))) /* A = 1.0 */
#define SIGMA_12_16QAM sqrt(2.50 * pow(10.0,(-EBNO / 10.0)))
#define SIGMA_34_16QAM sqrt((10.0/6.0) * pow(10.0,(-EBNO / 10.0)))

/* For 8AM, 64QAM, 256QAM, A = 0.5 => A*A = 0.25. Thus, Eav = 5.25*A*A = Eav/4 */
#define SIGMA_56_64QAM sqrt(4.2/4 * pow(10.0,(-EBNO / 10.0)))
#define SIGMA_46_64QAM sqrt(5.25/4 * pow(10.0,(-EBNO / 10.0)))
#define SIGMA_23_8AM sqrt(5.25/4 * pow(10.0,(-EBNO / 10.0)))
#define SIGMA_12_8AM sqrt(7.0/4 * pow(10.0,(-EBNO / 10.0)))
#define SIGMA_58_256QAM sqrt(17.0/4 * pow(10.0,(-EBNO / 10.0)))
#define SIGMA_68_256QAM sqrt((170.0/12)/4 * pow(10.0,(-EBNO / 10.0)))

/* For 4QAM (A = 0.5):*/
#define SIGMA_24_4QAM sqrt(2.0/2.0/4 * pow(10.0,(-EBNO / 10.0))) /* 1 info */
#define SIGMA_26_4QAM sqrt(2.0/(4.0/3)/4 * pow(10.0,(-EBNO / 10.0))) /* 2/3 info */
/* For 8QAM (A = 0.5):*/
#define SIGMA_4AM_of_46_8QAM sqrt(6.0/4.0/4 * pow(10.0,(-EBNO / 10.0))) /* 2 info*/
#define SIGMA_2AM_of_46_8QAM sqrt(6.0/4.0/4 * pow(10.0,(-EBNO / 10.0))) /* 2 info*/
#define SIGMA_4AM_of_26_8QAM sqrt(6.0/2.0/4 * pow(10.0,(-EBNO / 10.0))) /* 1 info*/
#define SIGMA_2AM_of_26_8QAM sqrt(6.0/2.0/4 * pow(10.0,(-EBNO / 10.0))) /* 1 info*/
#define SIGMA_4AM_of_13_8QAM sqrt(6.0/2.0/4 * pow(10.0,(-EBNO / 10.0))) /* 1 info*/
#define SIGMA_2AM_of_13_8QAM sqrt(6.0/2.0/4 * pow(10.0,(-EBNO / 10.0))) /* 1 info*/
#define SIGMA_36_64QAM sqrt(42.0/6.0/4 * pow(10.0,(-EBNO / 10.0))) /* 3 info */

/* For 16QAM, 64QAM, 256QAM, 1024QAM (A = 0.5):*/
#define SIGMA_412_16QAM sqrt(10.0/(8.0/3)/1 * pow(10.0,(-EBNO / 10.0))) /* 4/3 A=1 */
#define SIGMA_26_64QAM sqrt(42.0/4.0/4 * pow(10.0,(-EBNO / 10.0))) /* 2 info */
#define SIGMA_824_256QAM sqrt(170.0/(10.0/3)/4 * pow(10.0,(-EBNO / 10.0))) /* 10/3 */
#define SIGMA_1030_1024QAM sqrt(341.0/(10.0/3)/4 * pow(10.0,(-EBNO / 10.0))) /* 10/3 */

```

COMPUTER PROGRAM LISTING APPENDIX

```

/* For 32QAM, 128QAM, 512QAM, (A = 0.5):*/
#define SIGMA_8AM_of_115_32QAM  sqrt(21.0/2/4 * pow(10.0,(-EBNO / 10.0))) /* 1 info */
#define SIGMA_4AM_of_115_32QAM  sqrt(5.00/(4/3)/4 * pow(10.0,(-EBNO / 10.0))) /*2/3*/
#define SIGMA_16AM_of_721_128QAM  sqrt(85.0/(8/3)/4 * pow(10.0,(-EBNO / 10.0))) /*4/3*/
#define SIGMA_8AM_of_721_128QAM  sqrt(21.0/2/4 * pow(10.0,(-EBNO / 10.0))) /* 1 info */
#define SIGMA_32AM_of_39_512QAM  sqrt(341.0/4/4*pow(10.0,(-EBNO / 10.0))) /* 2 info */
#define SIGMA_16AM_of_39_512QAM  sqrt(85.00/2/4*pow(10.0,(-EBNO / 10.0))) /* 1 info */

/* For 32QAM (A = 0.5):*/
#define SIGMA_8AM_of_32QAM  sqrt(26.0/6/4 * pow(10.0,(-EBNO / 10.0))) /* 3 info */
#define SIGMA_4AM_of_32QAM  sqrt(26.0/6/4 * pow(10.0,(-EBNO / 10.0))) /* 3 info */

/* For R57_128QAM (A = 0.5):*/
#define SIGMA_16AM_of_128QAM  sqrt(106.0/10/4 * pow(10.0,(-EBNO / 10.0))) /* 3 info */
#define SIGMA_8AM_of_128QAM  sqrt(106.0/10/4 * pow(10.0,(-EBNO / 10.0))) /* 2 info */

/* For R69_512QAM (A = 0.5):*/
#define SIGMA_32AM_of_512QAM  sqrt(426.0/12/4*pow(10.0,(-EBNO / 10.0))) /* 4 info */
#define SIGMA_16AM_of_512QAM  sqrt(426.0/12/4*pow(10.0,(-EBNO / 10.0))) /* 2 info */

/* For R710_1024QAM (A = 0.5):*/
#define SIGMA_710_1024QAM  sqrt((341.0/7)/4 * pow(10.0,(-EBNO / 10.0))) /* 3.5 info */

/* Define the particular coding and modulation case for simulation */
#define R36_64QAM

#define BIT_HIST
#define THRESHOLD_ITER      10      /* record bit histogram for higher iterations */
#define MAX_BIT_HIST_ARRAY  (2 * INT_SIZE)
#define ERROR_FILE_NAME     "../results/R36_64QAM_6144_test_30.err"
#define FRAME_HIST_FILE_NAME "../results/test.fhist"
#define BIT_HIST_FILE_NAME   "../results/map.hist"
#define INTERLEAVER_FILE    "../results/6144/s6144"

/*
 * Note1:
Make sure that for each simulation, the INT_SIZE represents the size of the interleaver
defined in INTERLEAVER_FILE
 */

/*
 * Note2:
 *
In rate 4/6 64QAM_TTCM only two bits out of four are coded rate half. Therefore,
the first half of the interleaver table used has a INT_SIZE/2 interleaver,
the rest is mapping the bits in the same position.
 */

```

COMPUTER PROGRAM LISTING APPENDIX

Votc.c

```

/*
***** *****
* file:      tc_v2.c
* date:      March 20, 2000
* function:  tests turbo codes
*           Modulation: QAM
*           Decoder:    MAP algorithm
***** *****
*/
#include "tc_v2.h"

typedef struct {
    int      enc_state;          /* encoder state */          */
    int      nr_states;          /* number of encoder states */
    int      enc_mem;            /* encoder memory */          */
    int      bp;                 /* backward polynomial */     */
    int      fp;                 /* forward polynomial */     */
    int      *P0state;           /* previous state for i=0 branch */
    int      *P1state;           /* previous state for i=1 branch */
    int      *N0state;           /* next state for i=0 branch */
    int      *N1state;           /* next state for i=1 branch */
    int      *Coded0;             /* coded bit for i=0 branch */
    int      *Coded1;             /* coded bit for i=1 branch */
} jat_code;

void      jat_map1(jat_code *, double *, double *, double *);
void      jat_map2(jat_code *, double *, double *, double *);
void      jat_trellis_bp_fp(jat_code *);
int      jat_ps(jat_code *, int);
int      jat_enc_bp_fp(jat_code *, int);
void      r_ileav(double *, int *);
void      r_ileava(int *, int *);
void      r_deileav(double *, int *);
void      r_deileava(int *, int *);
double   nrgen();
int      nrgenbin();
double   gasdev();
int      errors(int *, double *, int, int);
int      print_err(int *, double *, int, int, int *);
int      find_tx_I(int);
double   find_tx_Q(int);

int      *frame_hist;          /* how many frames with how many errors*/
int      **bit_hist_array;     /* pointer to NR_ITER pointers
                                to blocks of data organised as:
                                block nr.,bit pos. in error,
                                block nr.,bit pos. in error */
int      *bit_hist_block;      /* current number of blocks in error
                                for each iteration */
int      frame_err;           /* frame/block error rate */
int      total_err;           /* total nr. of err. after NR_ITER */
long     s1, s2;               /* seed generators */

/*
***** *****
main()
{
jat_code  *jat_code1;
jat_code  *jat_code2;
int       u1, u2, u3, u4, u5, u6;    /* bits of a 64QAM symbol in TTCM */
double    tx_I, tx_Q, rx_I, rx_Q;
double    v00_I, v00_Q, v01_I, v01_Q, v10_I, v10_Q, v11_I, v11_Q;
int       i, j, k, block, iteration;
int       *rule;                  /* interleaver */
int       *data;                 /* the information block of data */
}

```

COMPUTER PROGRAM LISTING APPENDIX

```

int      *data_i;           /* the interleaved information block of data*/
int      *data_d;           /* the deinterleaved inf. block of data */
int      *Enc1;             /* Encoder1 output */
int      *Enc2;             /* Encoder2 output */
int      *no_err;            /* stores nr. err. for each iteration */
double   *D1_data;          /* Decoder1 input data */
double   *D1_parity;         /* Decoder1 input parity */
double   *D1_app;            /* Decoder1 input a priori information */
double   *D1_exi;            /* Decoder1 output extrinsic information */
double   *D2_data;          /* Decoder2 input data */
double   *D2_parity;         /* Decoder2 input parity */
double   *D2_app;            /* Decoder2 input a priori information */
double   *D2_exi;            /* Decoder2 output extrinsic information */
double   *Dec_data;          /* Decoded data */
double   *Zero_data;         /* zero data */
double   d0, d1, d2, d3, d4, L_d0, L_d1, L_d2, L_d3, tx, rx, K, noisel, n;
double   L_d4, L_d5;
double   L_u1, L_u2, L_u3, L_u4, L_u5, L_u6;
double   noise_I, noise_Q;
FILE    *out_file = NULL;

s1      = SEED1; /* initialize the seeds for the noise generator */ */
s2      = SEED2;
frame_err = 0;
total_err = 0;

/*
 * initialize the code structures:
 */
jat_code1      = (jat_code *)malloc(sizeof(jat_code));
jat_code1->enc_mem = RSC1_ENC_MEM;
jat_code1->bp  = RSC1_BP;
jat_code1->fp  = RSC1_FP;
jat_code1->enc_state = 0;
jat_code1->nr_states = (1 << RSC1_ENC_MEM);
jat_code1->P0state = (int *)malloc(sizeof(int)*jat_code1->nr_states);
jat_code1->P1state = (int *)malloc(sizeof(int)*jat_code1->nr_states);
jat_code1->N0state = (int *)malloc(sizeof(int)*jat_code1->nr_states);
jat_code1->N1state = (int *)malloc(sizeof(int)*jat_code1->nr_states);
jat_code1->Coded0 = (int *)malloc(sizeof(int)*jat_code1->nr_states);
jat_code1->Coded1 = (int *)malloc(sizeof(int)*jat_code1->nr_states);
jat_trellis_bp_fp(jat_code1);
jat_code2      = (jat_code *)malloc(sizeof(jat_code));
jat_code2->enc_mem = RSC2_ENC_MEM;
jat_code2->bp  = RSC2_BP;
jat_code2->fp  = RSC2_FP;
jat_code2->enc_state = 0;
jat_code2->nr_states = (1 << RSC2_ENC_MEM);
jat_code2->P0state = (int *)malloc(sizeof(int)*jat_code2->nr_states);
jat_code2->P1state = (int *)malloc(sizeof(int)*jat_code2->nr_states);
jat_code2->N0state = (int *)malloc(sizeof(int)*jat_code2->nr_states);
jat_code2->N1state = (int *)malloc(sizeof(int)*jat_code2->nr_states);
jat_code2->Coded0 = (int *)malloc(sizeof(int)*jat_code2->nr_states);
jat_code2->Coded1 = (int *)malloc(sizeof(int)*jat_code2->nr_states);
jat_trellis_bp_fp(jat_code2);

data = (int *)malloc(sizeof(int) * INT_SIZE);
if(data == 0)
{
    printf("Couldn't allocate data memory!\n");
    exit(1);
}

data_i = (int *)malloc(sizeof(int) * INT_SIZE);
if(data_i == 0)
{
    printf("Couldn't allocate data_i memory!\n");
    exit(1);
}

```

COMPUTER PROGRAM LISTING APPENDIX

```
data_d = (int *)malloc(sizeof(int) * INT_SIZE);
if(data_d == 0)
{
    printf("Couldn't allocate data_d memory!\n");
    exit(1);
}

no_err = (int *)malloc(sizeof(int) * NR_ITER);
if(no_err == 0)
{
    printf("Couldn't allocate no_err memory!\n");
    exit(1);
}
else
    for(i = 0; i < NR_ITER; i++)
        no_err[i] = 0;

Enc1 = (int *)malloc(sizeof(int) * INT_SIZE);
if(Enc1 == 0)
{
    printf("Couldn't allocate Enc1 memory!\n");
    exit(1);
}

Enc2 = (int *)malloc(sizeof(int) * INT_SIZE);
if(Enc2 == 0)
{
    printf("Couldn't allocate Enc2 memory!\n");
    exit(1);
}

D1_data = (double *)malloc(sizeof(double) * INT_SIZE);
if(D1_data == 0)
{
    printf("Couldn't allocate D1_data memory!\n");
    exit(1);
}

D1_parity = (double *)malloc(sizeof(double) * INT_SIZE);
if(D1_parity == 0)
{
    printf("Couldn't allocate D1_parity memory!\n");
    exit(1);
}

D1_app = (double *)malloc(sizeof(double) * INT_SIZE);
if(D1_app == 0)
{
    printf("Couldn't allocate D1_app memory!\n");
    exit(1);
}

D1_exi = (double *)malloc(sizeof(double) * INT_SIZE);
if(D1_exi == 0)
{
    printf("Couldn't allocate D1_exi memory!\n");
    exit(1);
}

D2_data = (double *)malloc(sizeof(double) * INT_SIZE);
if(D2_data == 0)
{
    printf("Couldn't allocate D2_data memory!\n");
    exit(1);
}

D2_parity = (double *)malloc(sizeof(double) * INT_SIZE);
if(D2_parity == 0)
{
    printf("Couldn't allocate D2_parity memory!\n");
    exit(1);
}
```

COMPUTER PROGRAM LISTING APPENDIX

```
}

D2_app = (double *)malloc(sizeof(double) * INT_SIZE);
if(D2_app == 0)
{
    printf("Couldn't allocate D2_app memory!\n");
    exit(1);
}

D2_exi = (double *)malloc(sizeof(double) * INT_SIZE);
if(D2_exi == 0)
{
    printf("Couldn't allocate D2_exi memory!\n");
    exit(1);
}

Dec_data = (double *)malloc(sizeof(double) * INT_SIZE);
if(Dec_data == 0)
{
    printf("Couldn't allocate Dec_data memory!\n");
    exit(1);
}

Zero_data = (double *)malloc(sizeof(double) * INT_SIZE);
if(Zero_data == 0)
{
    printf("Couldn't allocate Zero_data memory!\n");
    exit(1);
}
for(i = 0; i < INT_SIZE; i++)
    Zero_data[i] = 0.0;

frame_hist = (int *)malloc(sizeof(int) * (INT_SIZE+1) * NR_ITER);
if(frame_hist == 0)
{
    printf("Couldn't allocate frame_hist memory!\n");
    exit(1);
}
else
{
    for(i = 0; i < (INT_SIZE+1)*NR_ITER; i++)
        frame_hist[i] = 0;
}

bit_hist_array = (int **)malloc(sizeof(int *) * 2 * NR_ITER);
if(bit_hist_array == 0)
{
    printf("Couldn't allocate bit_hist_array memory!\n");
    exit(1);
}
else
{
    for(i = THRESHOLD_ITER; i <= NR_ITER; i++)
    {
        bit_hist_array[i] = (int *)malloc(sizeof(int) * MAX_BIT_HIST_ARRAY);
        bit_hist_array[i+NR_ITER] = bit_hist_array[i]; /* store the original pointer */
        if(bit_hist_array[i] == 0)
        {
            printf("Couldn't allocate bit_hist_array[i] memory!\n");
            exit(1);
        }
    }
}

bit_hist_block = (int *)malloc(sizeof(int) * (NR_ITER+1));
if(bit_hist_block == 0)
{
    printf("Couldn't allocate bit_hist_block memory!\n");
    exit(1);
}
else
```

COMPUTER PROGRAM LISTING APPENDIX

```

{
    for(i = 0; i <= NR_ITER; i++)
        bit_hist_block[i] = 1;
}

rule = (int *)malloc(sizeof(int) * INT_SIZE * 2);
if(rule == 0)
{
    printf("Couldn't allocate rule memory!\n");
    exit(1);
}

for(i = 0; i < INT_SIZE; i++)
    rule[2*i] = 0;

/*
 * read the interleaver file
 */

out_file = fopen(INTERLEAVER_FILE, "r");

if(!out_file)
{
    printf("Error1: the output file could not be opened!\n");
    exit (1);
}
for(i = 0; i < INT_SIZE; i++)
    fscanf(out_file, "%d%d", &i, &rule[2*i+1]);
fclose(out_file);

/*
 * initialize the noise generator seeds in order to have the same data and
 * noise for different random interleavers
 */
s1 = SEED1;
s2 = SEED2;

/*
 * start the big loop:
 */
for(block = 1; total_err < MAX_ERRORS; block++)
{
    jat_code1->enc_state = 0;                      /* reset encoder1's state */
    jat_code2->enc_state = 0;                      /* reset encoder2's state */
    for(i = 0; i < INT_SIZE; i++)                  /* no app for first decoder */
        D1_app[i] = 1.0;

    /*
     * generate random data:
     */
    for(i = 0; i < INT_SIZE; i++)
        data[i] = nrngenbin();

    /*
     * encoder1:
     */
    for(i = 0; i < INT_SIZE; i++)
        Enc1[i] = jat_enc_bp_fp(jat_code1, data[i]);

    /*
     * interleave data:
     */
    for(i = 0; i < INT_SIZE; i++)
        data_i[i] = data[i];
    r_ileava(data_i, rule);

    /*
     * encoder2:
     */
    for(i = 0; i < INT_SIZE; i++)

```

COMPUTER PROGRAM LISTING APPENDIX

```

Enc2[i] = jat_enc_bp_fp(jat_code2, data_i[i]);

/*
 * deinterleave data:
 */
for(i = 0; i < INT_SIZE; i++)
    data_d[i] = data[i];
r_deileava(data_d, rule);

/*
 * modulate and add AWGN noise:
 */
#endif R12_4AM
/*
 * Channel:
 * d0 is MSB and d1 is LSB in a 4-AM:(d0,d1) = 01--00-|-10--11
 *                                         -3   -1     1    3
 */
n = (-1.0) / (2 * SIGMA_12_4AM * SIGMA_12_4AM);
for(i = 0; i < INT_SIZE; i++)
{
    d0 = data[i];
    if(i & 0x1)
        d1 = Enc1[i];
    else
        d1 = Enc2[i];
    tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
    rx = tx + SIGMA_12_4AM * gasdev();
    L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /
                (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3)))); 
    L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3))) /
                (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1)))); 
    D1_data[i] = L_d0;
    if(i & 0x1)
    {
        D1_parity[i] = L_d1;
        D2_parity[i] = 0.0;
    }
    else
    {
        D1_parity[i] = 0.0;
        D2_parity[i] = L_d1;
    }
}
#endif

#ifndef R13_8AM
/*
 * Channel:
 * d0 is MSB and d2 is LSB in 8-AM:(d0,d1,d2):
 *          010---011---001---000---100---101---111---110
 *          -3.5   -2.5   -1.5   -0.5   0.5   1.5   2.5   3.5
 */
n = (-1.0) / (2 * SIGMA_13_8AM * SIGMA_13_8AM);
for(i = 0; i < INT_SIZE; i++)
{
    d0 = data[i];
    if(i & 0x1)
    {
        d1 = Enc1[i];
        d2 = Enc2[i];
    }
    else
    {
        d1 = Enc2[i];
        d2 = Enc1[i];
    }
}
#endif

```

COMPUTER PROGRAM LISTING APPENDIX

```

    }

tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
rx = tx + SIGMA_13_8AM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
            (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
            (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
            (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

D1_data[i] = L_d0;
if(i & 0x1)
{
    D1_parity[i] = L_d1;
    D2_parity[i] = L_d2;
}
else
{
    D1_parity[i] = L_d2;
    D2_parity[i] = L_d1;
}
}

#endif

#ifndef R12_8AM
/*
 * Channel:
 * d0 is MSB and d2 is LSB in 8-AM: (d0,d1,d2):
 *      010---011---001---000---100---101---111---110
 *      -3.5   -2.5   -1.5   -0.5    0.5    1.5    2.5    3.5
 */
/*
 * Channel: we transmit two 8AM symbols to emulate a 64QAM symbol.
 * 6 info bits and 6 parity bits are mapped to 2 64QAM symbols which in
 * turn are simulated as 4 8AM symbols to achieve 3bit/s/Hz
 *
 * INT_SIZE to be a multiple of 6
 */
n = (-1.0) / (2 * SIGMA_12_8AM * SIGMA_12_8AM);
for(i = 0; i < INT_SIZE; i++)
{
    /* symbol 1 */
    d0 = data[i];
    d1 = data[i+1];
    d2 = Enc1[i];
    tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
    rx = tx + SIGMA_12_8AM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

    L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
                (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
```

COMPUTER PROGRAM LISTING APPENDIX

```

D1_data[i]      = L_d0;
D1_data[i+1]    = L_d1;
D1_parity[i]    = L_d2;
D1_parity[i+1]  = 0;
D2_parity[i]    = 0;

/* symbol 2 */
d0   = data[i+2];
d1   = Enc1[i+2];
d2   = Enc2[i+1];
tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
rx   = tx + SIGMA_12_8AM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
            (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
            (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
            (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

D1_data[i+2]    = L_d0;
D1_parity[i+2]  = L_d1;
D2_parity[i+1]  = L_d2;
D2_parity[i+2]  = 0;

/* symbol 3 */
d0   = data[i+3];
d1   = data[i+4];
d2   = Enc2[i+3];
tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
rx   = tx + SIGMA_12_8AM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
            (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
            (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
            (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

D1_data[i+3]    = L_d0;
D1_data[i+4]    = L_d1;
D2_parity[i+3]  = L_d2;
D1_parity[i+3]  = 0;
D2_parity[i+4]  = 0;

/* symbol 4 */
d0   = data[i+5];
d1   = Enc1[i+4];
d2   = Enc2[i+5];
tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
rx   = tx + SIGMA_12_8AM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
            (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
            (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));


```

COMPUTER PROGRAM LISTING APPENDIX

```

        exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
(exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

D1_data[i+5] = L_d0;
D1_parity[i+4] = L_d1;
D2_parity[i+5] = L_d2;
D1_parity[i+5] = 0;

i = i+5;
}
#endif

#ifndef R23_8AM
/*
 * Channel:
 * d0 is MSB and d2 is LSB in 8-AM: (d0,d1,d2):
 *      010---011---001---000---100---101---111---110
 *      -3.5   -2.5   -1.5   -0.5    0.5    1.5    2.5    3.5
 */
n = (-1.0) / (2 * SIGMA_23_8AM * SIGMA_23_8AM);
for(i = 0; i < INT_SIZE; i++)
{
    d0 = data[i];
    d1 = data[i+1];
    if(i & 0x4)
        d2 = Enc1[i];
    else
        d2 = Enc2[i];

    tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
    rx = tx + SIGMA_23_8AM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
(exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

    L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
(exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

    D1_data[i] = L_d0;
    D1_data[i+1] = L_d1;
    if(i & 0x4)
    {
        D1_parity[i] = L_d2;
        D1_parity[i+1] = 0;
        D2_parity[i] = 0;
        D2_parity[i+1] = 0;
    }
    else
    {
        D1_parity[i] = 0;
        D1_parity[i+1] = 0;
        D2_parity[i] = L_d2;
        D2_parity[i+1] = 0;
    }
}

i++;
}
#endif

```

COMPUTER PROGRAM LISTING APPENDIX

```

#ifndef R35_32QAM
/*
 * I dimension:
 * d0 is MSB and d2 is LSB in 8-AM: (d0,d1,d2):
 *      010---011---001---000---100---101---111---110
 *      -3.5   -2.5   -1.5   -0.5    0.5    1.5    2.5    3.5
 * Q dimension:
 * d0 is MSB and d1 is LSB in a 4-AM: (d0,d1):
 *      01----00----10----11
 *      -1.5   -0.5   0.5    1.5
 *
 * We transmit one 8AM symbol and one 4AM symbol to emulate a 32QAM symbol.
 * 6 info bits and 4 parity bits are mapped to 2 32QAM symbols.
 *
 * INT_SIZE to be a multiple of 6
 */

for(i = 0; i < INT_SIZE; i++)
{
    /* symbol 1: 8AM */
    d0 = data[i];
    d1 = data[i+2];
    d2 = Enc1[i];
    tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
    rx = tx + SIGMA_8AM_of_32QAM * gasdev();
    n = (-1.0) / (2 * SIGMA_8AM_of_32QAM * SIGMA_8AM_of_32QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

    L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
                (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

    D1_data[i] = L_d0;
    D1_data[i+2] = L_d1;
    D1_parity[i] = L_d2;
    D1_parity[i+1] = 0.0;
    D1_parity[i+2] = 0.0;
    D2_parity[i] = 0.0;
    D2_parity[i+2] = 0.0;

    /* symbol 2: 4AM */
    d0 = data[i+1];
    d1 = Enc2[i+1];
    tx = d0 - d1 + 2*d0*d1 - 0.5;
    rx = tx + SIGMA_4AM_of_32QAM * gasdev();
    n = (-1.0) / (2 * SIGMA_4AM_of_32QAM * SIGMA_4AM_of_32QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) /
                (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5))));

    L_d1 = log((exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx-1.5)*(rx-1.5)) /
                (exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx+0.5)*(rx+0.5))));

    D1_data[i+1] = L_d0;
    D2_parity[i+1] = L_d1;

    /* symbol 3: 8AM */
    d0 = data[i+3];
    d1 = data[i+5];
    d2 = Enc2[i+4];
    tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
    rx = tx + SIGMA_8AM_of_32QAM * gasdev();
    n = (-1.0) / (2 * SIGMA_8AM_of_32QAM * SIGMA_8AM_of_32QAM);
}

```

COMPUTER PROGRAM LISTING APPENDIX

```

L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
            (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
            (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
            (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

D1_data[i+3] = L_d0;
D1_data[i+5] = L_d1;
D1_parity[i+4] = 0.0;
D1_parity[i+5] = 0.0;
D2_parity[i+3] = 0.0;
D2_parity[i+4] = L_d2;
D2_parity[i+5] = 0.0;

/* symbol 4: 4AM */
d0 = data[i+4];
d1 = Enc1[i+3];
tx = d0 - d1 + 2*d0*d1 - 0.5;
rx = tx + SIGMA_4AM_of_32QAM * gasdev();
n = (-1.0) / (2 * SIGMA_4AM_of_32QAM * SIGMA_4AM_of_32QAM);
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) /
            (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5))));
L_d1 = log((exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx-1.5)*(rx-1.5)) /
            (exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx+0.5)*(rx+0.5))));
D1_data[i+4] = L_d0;
D1_parity[i+3] = L_d1;

i = i+5;
}
#endif

/*
 * R46_64QAM_TTCM_VoCAL
 * Option 4
 * Channel: I & Q defined as
 *           -|-----|-----|-----|-----|-----|-----|- -
 *           -3.5   -2.5   -1.5   -0.5   0.5    1.5    2.5    3.5
 * the 64QAM symbol is defined as: (u1, u2, u3, u4, u5, u6)
 * where u4 = d0
 *       u3 = d1
 *       u2 = p0 parity from ENC_H
 *       u1 = q1 parity from ENC_V
 *       u5 = d uncoded
 *       u6 = d uncoded
 * Use s4096v interleaver - only first 2048 bits are interleaved
 */
n = (-1.0) / (2 * SIGMA_46_64QAM * SIGMA_46_64QAM);
for(i = 0; i < INT_SIZE/2 -1; )
{
    /* Encode only first half of INT_SIZE
     * d0, d1, d2, d3,...      up to INT_SIZE/2 - 1
     * p0, 0, p2, 0, ...
     * 0, q1, 0, q3, ...
     */
    u4 = data[i];
    u3 = data[i+1];
    u2 = Enc1[i];
    u1 = Enc2[i];
    u5 = data[i+INT_SIZE/2];
    u6 = data[i+INT_SIZE/2+1];

    k = u6+2*u5+4*u4+8*u3+16*u2+32*u1;
}

```

COMPUTER PROGRAM LISTING APPENDIX

COMPUTER PROGRAM LISTING APPENDIX

COMPUTER PROGRAM LISTING APPENDIX

```

exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+3.5)*(rx_Q+3.5))) /
(exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+3.5)*(rx_Q+3.5)));
L_d3 = log((exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +

```

COMPUTER PROGRAM LISTING APPENDIX

```

exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+3.5)*(rx_Q+3.5))) /
(exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+0.5)*(rx_Q+0.5)));
exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+0.5)*(rx_Q+0.5))));

D1_data[i]      = L_d3;
D1_data[i+1]    = L_d2;
D1_data[i + INT_SIZE/2] = rx_I;
D1_data[i + INT_SIZE/2 + 1] = rx_Q;
D1_parity[i]   = L_d1;
D1_parity[i+1] = 0;
D2_parity[i]   = 0;
D2_parity[i+1] = L_d0;

i = i + 2;
}
#endif

#ifndef R46_64QAM_TTCM_Ungerboeck_Map
/* Option3: conventional set partitioning used in TCM
 * Channel: I & Q defined
 *          -|----|----|----|----|----|----|----|-

```

COMPUTER PROGRAM LISTING APPENDIX

```

*      -3.5  -2.5  -1.5  -0.5   0.5   1.5   2.5   3.5
* the 64QAM symbol is defined as: (u1, u2, u3, u4, u5, u6)
* where
*      u6 = d0
*      u5 = d1
*      u4 = d2
*      u3 = d3
*      u2 = p0 parity from ENC_H
*      u1 = q1 parity from ENC_V
*/
/*
 * deinterleave data:
 */
for(i = 0; i < INT_SIZE; i++)
    data_d[i] = data[i];
r_deileava(data_d, rule);
n = (-1.0) / (2 * SIGMA_46_64QAM * SIGMA_46_64QAM);
for(i = 0; i < INT_SIZE; )
{
    /* Puncturing patern is:
     * d0, d1, d2, d3, ...
     * p0, 0, 0, 0, ...
     * 0, 0, q2, 0, ...
    */
    u1 = Enc1[i];
    u2 = Enc2[i];
    u6 = data_d[i+3];
    u5 = data_d[i+2];
    u4 = data_d[i+1];
    u3 = data_d[i];

    k = u6+2*u5+4*u4+8*u3+16*u2+32*u1;

    tx_I = find_tx_I(k);
    tx_Q = find_tx_Q(k);

    rx_I = tx_I + SIGMA_46_64QAM * gasdev();
    rx_Q = tx_Q + SIGMA_46_64QAM * gasdev();

    L_d0 = log((exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
                exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
                exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
                exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
                exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
                exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
                exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
                exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
                exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
                exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
                exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +

```

COMPUTER PROGRAM LISTING APPENDIX

COMPUTER PROGRAM LISTING APPENDIX

```

exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+3.5)*(rx_Q+3.5)));

```

COMPUTER PROGRAM LISTING APPENDIX

```

        exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
        exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
        exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+3.5)*(rx_Q+3.5))));

D1_data[i]      = L_d2;
D1_data[i+1]    = L_d3;
D1_data[i+2]    = L_d4;
D1_data[i+3]    = L_d5;
D1_parity[i]    = L_d0;
D1_parity[i+1]  = 0.0;
D1_parity[i+2]  = 0.0;
D1_parity[i+3]  = 0.0;
D2_parity[i]    = L_d1;
D2_parity[i+1]  = 0.0;
D2_parity[i+2]  = 0.0;
D2_parity[i+3]  = 0.0;

i = i + 4;
}
/*
 * interleave data:
 */
r_ileav(D1_data, rule);

#endif

#ifndef R46_64QAM_IQ_Natural_Map
/* Option2
 * Channel: I = (u1, u2, u3), Q = (u4, u5, u6) defined using natural mapping:
 *          000 001 010 011 100 101 110 111
 *          -|- -|- -|- -|- -|- -|
 *          -3.5 -2.5 -1.5 -0.5 0.5 1.5 2.5 3.5
 * the 64QAM symbol is defined as: (u1, u2, u3, u4, u5, u6)
 * where:
 *      u1 = d0
 *      u2 = d1
 *      u3 = p0 parity from ENC_V
 *      u4 = d2
 *      u5 = d3
 *      u6 = q0 parity from ENC_H
 *
 */
/*
 * deinterleave data:
 */
for(i = 0; i < INT_SIZE; i++)
    data_d[i] = data[i];
r_deileava(data_d, rule);

n = (-1.0) / (2 * SIGMA_46_64QAM * SIGMA_46_64QAM);

for(i = 0; i < INT_SIZE;)
{
    /* Puncturing patern is:
     * d0, d1, d2, d3, ...
     * p0, 0, 0, 0, ...
     * q0, 0, 0, 0, ...
     */
    u1 = data_d[i];
    u2 = data_d[i+1];
    u3 = Enc1[i];
    u4 = data_d[i+2];
    u5 = data_d[i+3];
    u6 = Enc2[i];

    tx_I = -3.5 + u3 + 2*u2 +4*u1;
    tx_Q = -3.5 + u6 + 2*u5 +4*u4;

    rx_I = tx_I + SIGMA_46_64QAM * gasdev();
}

```

COMPUTER PROGRAM LISTING APPENDIX

```

rx_Q = tx_Q + SIGMA_46_64QAM * gasdev();

L_u1 = log((exp(n*((rx_I-3.5)*(rx_I-3.5)))+
            exp(n*((rx_I-2.5)*(rx_I-2.5)))+
            exp(n*((rx_I-1.5)*(rx_I-1.5)))+
            exp(n*((rx_I-0.5)*(rx_I-0.5)))/
            (exp(n*((rx_I+3.5)*(rx_I+3.5)))+
            exp(n*((rx_I+2.5)*(rx_I+2.5)))+
            exp(n*((rx_I+1.5)*(rx_I+1.5)))+
            exp(n*((rx_I+0.5)*(rx_I+0.5)))));

L_u2 = log((exp(n*((rx_I-3.5)*(rx_I-3.5)))+
            exp(n*((rx_I-2.5)*(rx_I-2.5)))+
            exp(n*((rx_I+1.5)*(rx_I+1.5)))+
            exp(n*((rx_I+0.5)*(rx_I+0.5)))/
            (exp(n*((rx_I+3.5)*(rx_I+3.5)))+
            exp(n*((rx_I+2.5)*(rx_I+2.5)))+
            exp(n*((rx_I-1.5)*(rx_I-1.5)))+
            exp(n*((rx_I-0.5)*(rx_I-0.5)))));

L_u3 = log((exp(n*((rx_I-3.5)*(rx_I-3.5)))+
            exp(n*((rx_I+2.5)*(rx_I+2.5)))+
            exp(n*((rx_I-1.5)*(rx_I-1.5)))+
            exp(n*((rx_I+0.5)*(rx_I+0.5)))/
            (exp(n*((rx_I+3.5)*(rx_I+3.5)))+
            exp(n*((rx_I-2.5)*(rx_I-2.5)))+
            exp(n*((rx_I+1.5)*(rx_I+1.5)))+
            exp(n*((rx_I-0.5)*(rx_I-0.5))));

rx_I = rx_Q;

L_u4 = log((exp(n*((rx_I-3.5)*(rx_I-3.5)))+
            exp(n*((rx_I-2.5)*(rx_I-2.5)))+
            exp(n*((rx_I-1.5)*(rx_I-1.5)))+
            exp(n*((rx_I-0.5)*(rx_I-0.5)))/
            (exp(n*((rx_I+3.5)*(rx_I+3.5)))+
            exp(n*((rx_I+2.5)*(rx_I+2.5)))+
            exp(n*((rx_I+1.5)*(rx_I+1.5)))+
            exp(n*((rx_I+0.5)*(rx_I+0.5))));

L_u5 = log((exp(n*((rx_I-3.5)*(rx_I-3.5)))+
            exp(n*((rx_I-2.5)*(rx_I-2.5)))+
            exp(n*((rx_I+1.5)*(rx_I+1.5)))+
            exp(n*((rx_I+0.5)*(rx_I+0.5)))/
            (exp(n*((rx_I+3.5)*(rx_I+3.5)))+
            exp(n*((rx_I+2.5)*(rx_I+2.5)))+
            exp(n*((rx_I-1.5)*(rx_I-1.5)))+
            exp(n*((rx_I-0.5)*(rx_I-0.5))));

L_u6 = log((exp(n*((rx_I-3.5)*(rx_I-3.5)))+
            exp(n*((rx_I+2.5)*(rx_I+2.5)))+
            exp(n*((rx_I-1.5)*(rx_I-1.5)))+
            exp(n*((rx_I+0.5)*(rx_I+0.5)))/
            (exp(n*((rx_I+3.5)*(rx_I+3.5)))+
            exp(n*((rx_I-2.5)*(rx_I-2.5)))+
            exp(n*((rx_I+1.5)*(rx_I+1.5)))+
            exp(n*((rx_I-0.5)*(rx_I-0.5))));

D1_data[i]      = L_u1;
D1_data[i+1]    = L_u2;
D1_data[i+2]    = L_u4;
D1_data[i+3]    = L_u5;
D1_parity[i]    = L_u3;
D1_parity[i+1]  = 0.0;
D1_parity[i+2]  = 0.0;
D1_parity[i+3]  = 0.0;
D2_parity[i]    = L_u6;
D2_parity[i+1]  = 0.0;
D2_parity[i+2]  = 0.0;
D2_parity[i+3]  = 0.0;

i = i + 4;
}
/*
* interleave data:

```

COMPUTER PROGRAM LISTING APPENDIX

```

        */
r_ileav(D1_data, rule);

#endif

#ifndef R46_64QAM_IQ_Gray_Map
/* Option1: used in "Parallel Concatenated Trellis Coded Modulation" ICC'96
 * Channel: I = (u1, u2, u3), Q = (u4, u5, u6) defined using Gray mapping:
 * u1 & u4 are MSBs and u3 & u6 are LSBs:
 *      010---011---001---000---100---101---111---110
 *      -3.5   -2.5   -1.5   -0.5   0.5    1.5    2.5    3.5
 * where:
 *      u1 = d0
 *      u2 = d1
 *      u3 = p0 parity from ENC_V
 *      u4 = d2
 *      u5 = d3
 *      u6 = q0 parity from ENC_H
 *
 * INT_SIZE = multiple of 4
 */
n = (-1.0) / (2 * SIGMA_23_8AM * SIGMA_23_8AM);

/*
 * deinterleave data:
 */
for(i = 0; i < INT_SIZE; i++)
    data_d[i] = data[i];
r_deileava(data_d, rule);

for(i = 0; i < INT_SIZE;)
{
    /* Puncturing patern is:
     * d0, d1, d2, d3, ...
     * p0, 0, 0, 0, ...
     * q0, 0, 0, 0, ...
     */
    u1 = data_d[i];
    u2 = data_d[i+1];
    u3 = Enc1[i];
    tx = 2*u1 - 2*u2 + 4*u1*u2 - 1.0 + (((2*u1-1)*(2*u2-1))<0?(u3-0.5):(0.5-u3));
    rx = tx + SIGMA_23_8AM * gasdev();
    L_u1 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

    L_u2 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

    L_u3 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
                (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

    u1 = data_d[i+2];
    u2 = data_d[i+3];
    u3 = Enc2[i];
    tx = 2*u1 - 2*u2 + 4*u1*u2 - 1.0 + (((2*u1-1)*(2*u2-1))<0?(u3-0.5):(0.5-u3));
    rx = tx + SIGMA_23_8AM * gasdev();
    L_u4 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

    L_u5 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +

```

COMPUTER PROGRAM LISTING APPENDIX

```

exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
(exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
 exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

L_u6 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
 exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
(exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
 exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

D1_data[i]      = L_u1;
D1_data[i+1]    = L_u2;
D1_data[i+2]    = L_u4;
D1_data[i+3]    = L_u5;
D1_parity[i]   = L_u3;
D1_parity[i+1] = 0.0;
D1_parity[i+2] = 0.0;
D1_parity[i+3] = 0.0;
D2_parity[i]   = L_u6;
D2_parity[i+1] = 0.0;
D2_parity[i+2] = 0.0;
D2_parity[i+3] = 0.0;

i = i + 4;

}

/*
 * interleave data:
 */
r_ileav(D1_data, rule);

#endif

/*mio*/
#ifndef R24_4QAM
/*
 * Channel: we transmit two 2-AM symbols to emulate a 4-QAM symbol.
 * 2 info bits and 2 parity bits are mapped to 2 4-QAM symbols which in
 * turn are simulated as 4 2-AM symbols to achieve 1bit/s/Hz
 *
 * INT_SIZE to be a multiple of 2
 */

n = (-1.0) / (2 * SIGMA_24_4QAM * SIGMA_24_4QAM);
for(i = 0; i < INT_SIZE; i++)
{
    /* symbol 1 */
    d0 = data[i];
    tx = d0 - 0.5;
    rx = tx + SIGMA_24_4QAM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5)) /
                (exp(n*(rx+0.5)*(rx+0.5)))));

    D1_data[i] = L_d0;

    /* symbol 2 */
    d0 = Enc1[i];
    tx = d0 - 0.5;
    rx = tx + SIGMA_24_4QAM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5)) /
                (exp(n*(rx+0.5)*(rx+0.5)))));

    D1_parity[i] = L_d0;

    /* symbol 3 */
    d0 = data[i+1];
    tx = d0 - 0.5;
    rx = tx + SIGMA_24_4QAM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5)) /
                (exp(n*(rx+0.5)*(rx+0.5)))));

    D1_data[i+1] = L_d0;

    /* symbol 4 */
}

```

COMPUTER PROGRAM LISTING APPENDIX

```

d0    = Enc2[i+1];
tx    = d0 - 0.5;
rx    = tx + SIGMA_24_4QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
            (exp(n*(rx+0.5)*(rx+0.5)))); 
D2_parity[i+1] = L_d0;
D1_parity[i+1] = 0.0;
D2_parity[i]   = 0.0;
i = i+1;
}
#endif

/*mio*/
#ifndef R26_4QAM
/*
 * Channel: we transmit two 2-AM symbols to emulate a 4-QAM symbol.
 * 2 info bits and 4 parity bits are mapped to 3 4-QAM symbols which in
 * turn are simulated as 6 2-AM symbols to achieve 1bit/s/Hz
 *
 * INT_SIZE to be a multiple of 2
 */
n    = (-1.0) / (2 * SIGMA_26_4QAM * SIGMA_26_4QAM);
for(i = 0; i < INT_SIZE; i++)
{
    /* symbol 1 */
    d0    = data[i];
    tx    = d0 - 0.5;
    rx    = tx + SIGMA_26_4QAM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
                (exp(n*(rx+0.5)*(rx+0.5)))); 
    D1_data[i] = L_d0;

    /* symbol 2 */
    d0    = Enc1[i];
    tx    = d0 - 0.5;
    rx    = tx + SIGMA_26_4QAM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
                (exp(n*(rx+0.5)*(rx+0.5)))); 
    D1_parity[i] = L_d0;

    /* symbol 3 */
    d0    = Enc2[i];
    tx    = d0 - 0.5;
    rx    = tx + SIGMA_26_4QAM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
                (exp(n*(rx+0.5)*(rx+0.5)))); 
    D2_parity[i] = L_d0;

    /* symbol 4 */
    d0    = data[i+1];
    tx    = d0 - 0.5;
    rx    = tx + SIGMA_26_4QAM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
                (exp(n*(rx+0.5)*(rx+0.5)))); 
    D1_data[i+1] = L_d0;

    /* symbol 5 */
    d0    = Enc1[i+1];
    tx    = d0 - 0.5;
    rx    = tx + SIGMA_26_4QAM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
                (exp(n*(rx+0.5)*(rx+0.5)))); 
    D1_parity[i+1] = L_d0;

    /* symbol 6 */
    d0    = Enc2[i+1];
    tx    = d0 - 0.5;
    rx    = tx + SIGMA_26_4QAM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
                (exp(n*(rx+0.5)*(rx+0.5)))); 
}

```

COMPUTER PROGRAM LISTING APPENDIX

```

D2_parity[i+1] = L_d0;
i = i+1;
}
#endif

/*mio*/
#ifndef R46_8QAM
/*
 * I dimension:
 * d0 is MSB and d1 is LSB in a 4-AM: (d0,d1):
 *      01----00----10----11
 *      -1.5   -0.5   0.5   1.5
 * Q dimension:
 * d0 is MSB in a 2-AM: (d0):
 *      0-----1
 *      -0.5   0.5
 *
 *
 * We transmit one 4A-M symbol and one 2-AM symbol to emulate a 32QAM symbol.
 * 4 info bits and 2 parity bits are mapped to 2 8QAM symbols.
 *
 * INT_SIZE to be a multiple of 4
*/
for(i = 0; i < INT_SIZE; i++)
{
    /* symbol 1: 4AM */
    d0 = data[i];
    d1 = Enc1[i];
    tx = d0 - d1 + 2*d0*d1 - 0.5;
    rx = tx + SIGMA_4AM_of_46_8QAM * gasdev();
    n = (-1.0) / (2 * SIGMA_4AM_of_46_8QAM * SIGMA_4AM_of_46_8QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))) /
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5))));
    L_d1 = log((exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx-1.5)*(rx-1.5))) /
               (exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx+0.5)*(rx+0.5))));
    D1_data[i] = L_d0;
    D1_parity[i] = L_d1;

    /* symbol 2: 2AM */
    d0 = data[i+1];
    tx = d0 - 0.5;
    rx = tx + SIGMA_2AM_of_46_8QAM * gasdev();
    n = (-1.0) / (2 * SIGMA_2AM_of_46_8QAM * SIGMA_2AM_of_46_8QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
               (exp(n*(rx+0.5)*(rx+0.5))));
    D1_data[i+1] = L_d0;

    /* symbol 3: 4AM */
    d0 = data[i+2];
    d1 = Enc2[i+2];
    tx = d0 - d1 + 2*d0*d1 - 0.5;
    rx = tx + SIGMA_4AM_of_46_8QAM * gasdev();
    n = (-1.0) / (2 * SIGMA_4AM_of_46_8QAM * SIGMA_4AM_of_46_8QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))) /
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5))));
    L_d1 = log((exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx-1.5)*(rx-1.5))) /
               (exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx+0.5)*(rx+0.5))));
    D1_data[i+2] = L_d0;
    D2_parity[i+2] = L_d1;

    /* symbol 2: 4AM */
    d0 = data[i+3];
    tx = d0 - 0.5;
    rx = tx + SIGMA_2AM_of_46_8QAM * gasdev();
    n = (-1.0) / (2 * SIGMA_2AM_of_46_8QAM * SIGMA_2AM_of_46_8QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
               (exp(n*(rx+0.5)*(rx+0.5))));
    D1_data[i+3] = L_d0;
    D1_parity[i+1] = 0.0;
    D1_parity[i+2] = 0.0;
}

```

COMPUTER PROGRAM LISTING APPENDIX

```

D1_parity[i+3] = 0.0;
D2_parity[i] = 0.0;
D2_parity[i+1] = 0.0;
D2_parity[i+3] = 0.0;

    i = i+3;
}
#endif

/*mio*/
#ifndef R26_8QAM
/*
 * I dimension:
 * d0 is MSB and d1 is LSB in a 4-AM: (d0,d1):
 *      01----00----10----11
 *      -1.5   -0.5   0.5   1.5
 * Q dimension:
 * d0 is MSB in a 2-AM: (d0):
 *      0-----1
 *      -0.5   0.5
 *
 *
 * We transmit one 4A-M symbol and one 2-AM symbol to emulate a 8QAM symbol.
 * 2 info bits and 4 parity bits are mapped to 2 8QAM symbols.
 *
 * INT_SIZE to be a multiple of 2
 */
for(i = 0; i < INT_SIZE; i++)
{
/* symbol 1: 4AM */

    d0 = data[i];
    d1 = Enc1[i];
    tx = d0 - d1 + 2*d0*d1 - 0.5;
    rx = tx + SIGMA_4AM_of_26_8QAM * gasdev();
    n = (-1.0) / (2 * SIGMA_4AM_of_26_8QAM * SIGMA_4AM_of_26_8QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))) /
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5))));
    L_d1 = log((exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx-1.5)*(rx-1.5))) /
               (exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx+0.5)*(rx+0.5))));
    D1_data[i] = L_d0;
    D1_parity[i] = L_d1;

/* symbol 2: 2AM */

    d0 = Enc2[i];
    tx = d0 - 0.5;
    rx = tx + SIGMA_2AM_of_26_8QAM * gasdev();
    n = (-1.0) / (2 * SIGMA_2AM_of_26_8QAM * SIGMA_2AM_of_26_8QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+
                 (exp(n*(rx+0.5)*(rx+0.5))));;
    D2_parity[i] = L_d0;

/* symbol 3: 4AM */

    d0 = Enc1[i+1];
    d1 = Enc2[i+1];
    tx = d0 - d1 + 2*d0*d1 - 0.5;
    rx = tx + SIGMA_4AM_of_26_8QAM * gasdev();
    n = (-1.0) / (2 * SIGMA_4AM_of_26_8QAM * SIGMA_4AM_of_26_8QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))) /
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5))));
    L_d1 = log((exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx-1.5)*(rx-1.5))) /
               (exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx+0.5)*(rx+0.5))));
    D1_parity[i+1] = L_d0;
    D2_parity[i+1] = L_d1;

/* symbol 4: 2AM */

    d0 = data[i+1];
}

```

COMPUTER PROGRAM LISTING APPENDIX

```

tx    = d0 - 0.5;
rx    = tx + SIGMA_2AM_of_26_8QAM * gasdev();
n    = (-1.0) / (2 * SIGMA_2AM_of_26_8QAM * SIGMA_2AM_of_26_8QAM);
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5)) /
            (exp(n*(rx+0.5)*(rx+0.5))));;
D1_data[i+1] = L_d0;
i=i+1;
}
#endif

/*mio*/
#ifndef R13_8QAM
/*
 * I dimension:
 * d0 is MSB and d1 is LSB in a 4-AM:(d0,d1):
 *      01----00----10----11
 *      -1.5   -0.5   0.5   1.5
 * Q dimension:
 * d0 is MSB in a 2-AM:(d0):
 *      0-----1
 *      -0.5   0.5
 *
 * We transmit one 4A-M symbol and one 2-AM symbol to emulate a 8QAM symbol.
 * 1 info bits and 2 parity bits are mapped to 1 8QAM symbols.
 *
 * INT_SIZE to be a multiple of 1
 */
for(i = 0; i < INT_SIZE; i++)
{
/* symbol 1: 4AM */

    d0 = data[i];
    d1 = Enc1[i];
    tx    = d0 - d1 + 2*d0*d1 - 0.5;
    rx    = tx + SIGMA_4AM_of_13_8QAM * gasdev();
    n    = (-1.0) / (2 * SIGMA_4AM_of_13_8QAM * SIGMA_4AM_of_13_8QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) /
                (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5))));;
    L_d1 = log((exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx-1.5)*(rx-1.5)) /
                (exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx+0.5)*(rx+0.5))));;
    D1_data[i] = L_d0;
    D1_parity[i] = L_d1;

/* symbol 2: 2AM */

    d0 = Enc2[i];
    tx    = d0 - 0.5;
    rx    = tx + SIGMA_2AM_of_13_8QAM * gasdev();
    n    = (-1.0) / (2 * SIGMA_2AM_of_13_8QAM * SIGMA_2AM_of_13_8QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5)) /
                (exp(n*(rx+0.5)*(rx+0.5))));;
    D2_parity[i] = L_d0;

}
#endif

/*mio*/
#ifndef R412_16QAM
/*
 * Channel: we transmit two 4-AM symbols to emulate a 16-QAM symbol.
 * 4 info bits and 8 parity bits are mapped to 3 16-QAM symbols which in
 * turn are simulated as 6 4-AM symbols to achieve 3bit/s/Hz
 * d0 is MSB and d1 is LSB in a 4-AM:(d0,d1) = 01----00--|---10----11
 *                                              -1.5   -0.5   0.5   1.5
 * INT_SIZE to be a multiple of 4
 */
n    = (-1.0) / (2 * SIGMA_412_16QAM * SIGMA_412_16QAM);
for(i = 0; i < INT_SIZE; i++)

```

COMPUTER PROGRAM LISTING APPENDIX

```

{
/* symbol 1 */
d0 = data[i];
d1 = Enc1[i];
tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
rx = tx + SIGMA_412_16QAM * gasdev();
L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /
            (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3)))); 
L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3))) /
            (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1)))); 
D1_data[i] = L_d0;
D1_parity[i] = L_d1;

/* symbol 2 */
d0 = data[i+1];
d1 = Enc1[i+1];
tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
rx = tx + SIGMA_412_16QAM * gasdev();
L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /
            (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3)))); 
L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3))) /
            (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1)))); 
D1_data[i+1] = L_d0;
D1_parity[i+1] = L_d1;

/* symbol 3 */
d0 = Enc2[i];
d1 = Enc2[i+1];
tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
rx = tx + SIGMA_412_16QAM * gasdev();
L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /
            (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3)))); 
L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3))) /
            (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1)))); 
D2_parity[i] = L_d0;
D2_parity[i+1] = L_d1;

/* symbol 4 */
d0 = data[i+2];
d1 = Enc1[i+2];
tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
rx = tx + SIGMA_412_16QAM * gasdev();
L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /
            (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3)))); 
L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3))) /
            (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1)))); 
D1_data[i+2] = L_d0;
D1_parity[i+2] = L_d1;

/* symbol 5 */
d0 = Enc2[i+2];
d1 = Enc2[i+3];
tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
rx = tx + SIGMA_412_16QAM * gasdev();
L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /
            (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3)))); 
L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3))) /
            (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1)))); 
D2_parity[i+2] = L_d0;
D2_parity[i+3] = L_d1;

/* symbol 6 */
d0 = data[i+3];
d1 = Enc1[i+3];
tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
rx = tx + SIGMA_412_16QAM * gasdev();
L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /
            (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3)))); 
L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3))) /
            (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1)))); 
D1_data[i+3] = L_d0;

```

COMPUTER PROGRAM LISTING APPENDIX

```

D1_parity[i+3] = L_d1;
    i = i+3;
}
#endif

/*mio*/
#ifndef R515_32QAM
/*
 * I dimension:
 * d0 is MSB and d2 is LSB in 8-AM:(d0,d1,d2):
 *      010---011---001---000---100---101---111---110
 *      -3.5   -2.5   -1.5   -0.5    0.5    1.5    2.5    3.5
 * Q dimension:
 * d0 is MSB and d1 is LSB in a 4-AM:(d0,d1):
 *      01----00----10----11
 *      -1.5   -0.5    0.5    1.5
 *
 *
 * We transmit one 8AM symbol and one 4AM symbol to emulate a 32QAM symbol.
 * 5 info bits and 10 parity bits are mapped to 3 32QAM symbols.
 *
 * INT_SIZE to be a multiple of 5
 */

for(i = 0; i < INT_SIZE; i++)
{
    /* symbol 1: 8AM */
    d0 = data[i];
    d1 = Enc1[i];
    d2 = Enc2[i];
    tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
    rx = tx + SIGMA_8AM_of_515_32QAM * gasdev();
    n = (-1.0) / (2 * SIGMA_8AM_of_515_32QAM * SIGMA_8AM_of_515_32QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

    L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
                (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

    D1_data[i] = L_d0;
    D1_parity[i] = L_d1;
    D2_parity[i] = L_d2;

    /* symbol 2: 4AM */
    d0 = data[i+1];
    d1 = Enc1[i+1];
    tx = d0 - d1 + 2*d0*d1 - 0.5;
    rx = tx + SIGMA_4AM_of_515_32QAM * gasdev();
    n = (-1.0) / (2 * SIGMA_4AM_of_515_32QAM * SIGMA_4AM_of_515_32QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) /
                (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5))));

    L_d1 = log((exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx-1.5)*(rx-1.5)) /
                (exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx+0.5)*(rx+0.5))));

    D1_data[i+1] = L_d0;
    D2_parity[i+1] = L_d1;

    /* symbol 3: 8AM */
    d0 = data[i+2];
    d1 = Enc1[i+2];
    d2 = Enc2[i+1];
    tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
    rx = tx + SIGMA_8AM_of_515_32QAM * gasdev();
}

```

COMPUTER PROGRAM LISTING APPENDIX

```

n    = (-1.0) / (2 * SIGMA_8AM_of_515_32QAM * SIGMA_8AM_of_515_32QAM);
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
            (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
            (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
            (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

D1_data[i+2] = L_d0;
D1_parity[i+2] = L_d1;
D2_parity[i+1] = L_d2;

/* symbol 4: 4AM */
d0 = data[i+3];
d1 = Enc2[i+2];
tx = d0 - d1 + 2*d0*d1 - 0.5;
rx = tx + SIGMA_4AM_of_515_32QAM * gasdev();
n = (-1.0) / (2 * SIGMA_4AM_of_515_32QAM * SIGMA_4AM_of_515_32QAM);
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5))));

L_d1 = log((exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx-1.5)*(rx-1.5)))/
            (exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx+0.5)*(rx+0.5))));

D1_data[i+3] = L_d0;
D2_parity[i+2] = L_d1;

/* symbol 5: 8AM */
d0 = data[i+4];
d1 = Enc1[i+3];
d2 = Enc2[i+3];
tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
rx = tx + SIGMA_8AM_of_515_32QAM * gasdev();
n = (-1.0) / (2 * SIGMA_8AM_of_515_32QAM * SIGMA_8AM_of_515_32QAM);
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
            (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
            (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
            (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

D1_data[i+4] = L_d0;
D1_parity[i+3] = L_d1;
D2_parity[i+3] = L_d2;

/* symbol 4: 4AM */
d0 = Enc1[i+4];
d1 = Enc2[i+4];
tx = d0 - d1 + 2*d0*d1 - 0.5;
rx = tx + SIGMA_4AM_of_515_32QAM * gasdev();
n = (-1.0) / (2 * SIGMA_4AM_of_515_32QAM * SIGMA_4AM_of_515_32QAM);
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) /
            (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5))));

L_d1 = log((exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx-1.5)*(rx-1.5)))/
            (exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx+0.5)*(rx+0.5))));

D1_parity[i+4] = L_d0;
D2_parity[i+4] = L_d1;

i = i+4;

```

COMPUTER PROGRAM LISTING APPENDIX

```

        }

#endif

/*mio*/
#ifndef R26_64QAM
/*
 * Channel:
 * d0 is MSB and d2 is LSB in 8AM:(d0,d1,d2):
 *      010---011---001---000---100---101---111---110
 *      -3.5   -2.5   -1.5   -0.5    0.5    1.5    2.5    3.5
 */
/*
 * Channel: we transmit two 8AM symbols to emulate a 64QAM symbol.
 * 2 info bits and 4 parity bits are mapped to 1 64QAM symbols which in
 * turn are simulated as 2 8AM symbols to achieve 2bit/s/Hz.
 *
 * INT_SIZE to be a multiple of 2
 */
n = (-1.0) / (2 * SIGMA_26_64QAM * SIGMA_26_64QAM);
for(i = 0; i < INT_SIZE; i++)
{
/* symbol 1 */

    d0 = data[i];
    d1 = Enc1[i];
    d2 = Enc2[i];
    tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
    rx = tx + SIGMA_26_64QAM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));
    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));
    L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
                (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
    D1_data[i] = L_d0;
    D1_parity[i] = L_d1;
    D2_parity[i] = L_d2;

/* symbol 2 */

    d0 = data[i+1];
    d1 = Enc1[i+1];
    d2 = Enc2[i+1];
    tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
    rx = tx + SIGMA_26_64QAM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));
    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));
    L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
                (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
    D1_data[i+1] = L_d0;
    D1_parity[i+1] = L_d1;
    D2_parity[i+1] = L_d2;
    i = i+1;
}

```

COMPUTER PROGRAM LISTING APPENDIX

```

        }

#endif

/*mio*/
#ifndef R36_64QAM
/*
 * Channel:
 * d0 is MSB and d2 is LSB in 8AM: (d0,d1,d2):
 *      010---011---001---000---100---101---111---110
 *      -3.5   -2.5   -1.5   -0.5    0.5    1.5    2.5    3.5
 */
/*
 * Channel: we transmit two 8AM symbols to emulate a 64QAM symbol.
 * 6 info bits and 6 parity bits are mapped to 2 64QAM symbols which in
 * turn are simulated as 4 8AM symbols to achieve 2bit/s/Hz.
 *
 * INT_SIZE to be a multiple of 6
 */
n = (-1.0) / (2 * SIGMA_36_64QAM * SIGMA_36_64QAM);
for(i = 0; i < INT_SIZE; i++)
{
/* symbol 1 */

    d0 = data[i];
    d1 = data[i+1];
    d2 = Enc1[i];
    tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
    rx = tx + SIGMA_36_64QAM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

    L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
                (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

    D1_data[i] = L_d0;
    D1_data[i+1] = L_d1;
    D1_parity[i] = L_d2;

/* symbol 2 */

    d0 = data[i+2];
    d1 = Enc1[i+2];
    d2 = Enc2[i+1];
    tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
    rx = tx + SIGMA_36_64QAM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

    L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
                (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

    D1_data[i+2] = L_d0;
    D1_parity[i+2] = L_d1;
    D2_parity[i+1] = L_d2;
}

```

COMPUTER PROGRAM LISTING APPENDIX

```

/* symbol 3 */

d0 = data[i+3];
d1 = data[i+4];
d2 = Enc2[i+3];
tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
rx = tx + SIGMA_36_64QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
(exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
(exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

D1_data[i+3] = L_d0;
D1_data[i+4] = L_d1;
D2_parity[i+3] = L_d2;

/* symbol 2 */

d0 = data[i+5];
d1 = Enc1[i+4];
d2 = Enc2[i+5];
tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
rx = tx + SIGMA_36_64QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
(exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
(exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

D1_data[i+5] = L_d0;
D1_parity[i+4] = L_d1;
D2_parity[i+5] = L_d2;
D2_parity[i] = 0.0;
D1_parity[i+1] = 0.0;
D2_parity[i+2] = 0.0;
D1_parity[i+3] = 0.0;
D2_parity[i+4] = 0.0;
D1_parity[i+5] = 0.0;
i = i+5;
}

#endif

/*mio*/
#ifndef R721_128QAM
/*
 * Q dimension:
 * d0 is MSB and d2 is LSB in 8-AM: (d0,d1,d2):
 * 010---011---001---000---100---101---111---110
 * -3.5 -2.5 -1.5 -0.5 0.5 1.5 2.5 3.5
 * I dimension:
 * d0 is MSB and d3 is LSB in 16AM: (d0,d1,d2,d3):
 *
 * 0010---0011---0001---0000---0100---0101---0111---0110
 * -7.5 -6.5 -5.5 -4.5 -3.5 -2.5 -1.5 -0.5
 */

```

COMPUTER PROGRAM LISTING APPENDIX

COMPUTER PROGRAM LISTING APPENDIX

```

exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
(exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i]      = L_d0;
D1_data[i+1]    = L_d1;
D1_parity[i]    = L_d2;
D2_parity[i]    = L_d3;

/* symbol 3 Q dimension: 8AM */

d0 = data[i+4];
d1 = Enc1[i+4];
d2 = Enc2[i+3];
tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
rx = tx + SIGMA_8AM_of_721_128QAM * gasdev();
n = (-1.0) / (2 * SIGMA_8AM_of_721_128QAM * SIGMA_8AM_of_721_128QAM);
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
(exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
(exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

D1_data[i+4]    = L_d0;
D1_parity[i+4]  = L_d1;
D2_parity[i+3]  = L_d2;

/* symbol 4 I dimension: 16AM */

d0 = data[i+3];
d1 = Enc1[i+3];
d2 = Enc1[i+2];
d3 = Enc2[i+2];
tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
tx = (d0 == 0 ? (tx - 4) : (4 - tx));
rx = tx + SIGMA_16AM_of_721_128QAM * gasdev();
n = (-1.0) / (2 * SIGMA_16AM_of_721_128QAM * SIGMA_16AM_of_721_128QAM);
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
(exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +

```

COMPUTER PROGRAM LISTING APPENDIX

```

exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
(exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i+3]      = L_d0;
D1_parity[i+3]    = L_d1;
D1_parity[i+2]    = L_d2;
D2_parity[i+2]    = L_d3;

/* symbol 5 Q dimension: 8AM */

d0 = data[i+6];
d1 = Enc1[i+6];
d2 = Enc2[i+6];
tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
rx = tx + SIGMA_8AM_of_721_128QAM * gasdev();
n = (-1.0) / (2 * SIGMA_8AM_of_721_128QAM * SIGMA_8AM_of_721_128QAM);
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
(exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
(exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

D1_data[i+6]      = L_d0;
D1_parity[i+6]    = L_d1;
D2_parity[i+6]    = L_d2;

/* symbol 6 I dimension: 16AM */

d0 = data[i+5];
d1 = Enc1[i+5];
d2 = Enc2[i+5];
d3 = Enc2[i+4];
tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
tx = (d0 == 0 ? (tx - 4) : (4 - tx));
rx = tx + SIGMA_16AM_of_721_128QAM * gasdev();
n = (-1.0) / (2 * SIGMA_16AM_of_721_128QAM * SIGMA_16AM_of_721_128QAM);
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/

```

COMPUTER PROGRAM LISTING APPENDIX

```

(exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
 exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
 exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
 exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
 exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
 exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
 exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
 (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
 exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
 exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
 exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
 exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
 exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
 exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
 (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
 exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
 exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
 exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i+5]      = L_d0;
D1_parity[i+5]    = L_d1;
D2_parity[i+5]    = L_d2;
D2_parity[i+4]    = L_d3;
i = i+6;
}
#endif

/*mio*/
#ifndef R824_256QAM
/*
 * Channel:
 * d0 is MSB and d3 is LSB in 16AM:(d0,d1,d2,d3):
 *
 *      0010---0011---0001---0000---0100---0101---0111---0110
 *      -7.5   -6.5   -5.5   -4.5   -3.5   -2.5   -1.5   -0.5
 *
 *      1010---1011---1001---1000---1100---1101---1111---1110
 *      7.5    6.5    5.5    4.5    3.5    2.5    1.5    0.5
 *
 * Channel: we transmit two 16AM symbols to emulate a 256QAM symbol.
 * 8 info bits and 16 parity bits are mapped to 3 256QAM symbols which in
 * turn are simulated as 6 16AM symbols to achieve 8/3bit/s/Hz.
 *
 * INT_SIZE to be a multiple of 8
 */
n    = (-1.0) / (2 * SIGMA_824_256QAM * SIGMA_824_256QAM);

/*
 * deinterleave data:
 */
for(i = 0; i < INT_SIZE; i++)
    data_d[i] = data[i];
r_deileava(data_d, rule);

for(i = 0; i < INT_SIZE; i++)
{
    /* symbol 1 */
    d0 = data_d[i];
    d1 = data_d[i+1];
    d2 = Enc1[i];
    d3 = Enc2[i];
    tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
    tx = (d0 == 0 ? (tx - 4): (4 - tx));
    /* Test the mapping to the 16AM constellation:
     * if (i < 500)

```

COMPUTER PROGRAM LISTING APPENDIX

```

*   printf("\n%d%d%d%d = %f", (int)d0, (int)d1, (int)d2, (int)d3, tx);
*/
rx = tx + SIGMA_58_256QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
(exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
(exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i] = L_d0;
D1_data[i+1] = L_d1;
D1_parity[i] = L_d2;
D2_parity[i] = L_d3;

/* symbol 2 */
d0 = data_d[i+2];
d1 = Enc1[i+2];
d2 = Enc1[i+1];
d3 = Enc2[i+1];
tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + ((2*d1-1)*(2*d2-1)<0?(d3-0.5):(0.5-d3));
tx = (d0 == 0 ? (tx - 4) : (4 - tx));
rx = tx + SIGMA_824_256QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
(exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +

```

COMPUTER PROGRAM LISTING APPENDIX

```

exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
(exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i+2] = L_d0;
D1_parity[i+2] = L_d1;
D1_parity[i+1] = L_d2;
D2_parity[i+1] = L_d3;

/* symbol 3 */
d0 = data_d[i+3];
d1 = Enc1[i+3];
d2 = Enc2[i+3];
d3 = Enc1[i+2];
tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
tx = (d0 == 0 ? (tx - 4) : (4 - tx));
rx = tx + SIGMA_824_256QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
(exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
(exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i+3] = L_d0;
D1_parity[i+3] = L_d1;
D2_parity[i+3] = L_d2;
D2_parity[i+2] = L_d3;

/* symbol 4 */
d0 = data_d[i+4];

```

COMPUTER PROGRAM LISTING APPENDIX

```

d1 = data_d[i+5];
d2 = Enc1[i+4];
d3 = Enc2[i+4];
tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
tx = (d0 == 0 ? (tx - 4): (4 - tx));
rx = tx + SIGMA_824_256QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
(exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
(exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i+4] = L_d0;
D1_data[i+5] = L_d1;
D1_parity[i+4] = L_d2;
D2_parity[i+4] = L_d3;

/* symbol 5 */

d0 = data_d[i+6];
d1 = Enc1[i+6];
d2 = Enc1[i+5];
d3 = Enc2[i+5];
tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
tx = (d0 == 0 ? (tx - 4): (4 - tx));
rx = tx + SIGMA_824_256QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
(exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));
```

COMPUTER PROGRAM LISTING APPENDIX

```

exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
(exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i+6] = L_d0;
D1_parity[i+6] = L_d1;
D1_parity[i+5] = L_d2;
D2_parity[i+5] = L_d3;

/* symbol 4 */
d0 = data_d[i+7];
d1 = Enc1[i+7];
d2 = Enc2[i+7];
d3 = Enc2[i+6];
tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
tx = (d0 == 0 ? (tx - 4) : (4 - tx));
rx = tx + SIGMA_824_256QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
(exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
(exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i+7] = L_d0;
D1_parity[i+7] = L_d1;
D2_parity[i+7] = L_d2;
D2_parity[i+6] = L_d3;

```

COMPUTER PROGRAM LISTING APPENDIX

COMPUTER PROGRAM LISTING APPENDIX

```

exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5))),;

L_d1 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)))/
(exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)) +
exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)) +
exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5))),;

L_d2 = log((exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5))),;

L_d3 = log((exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)))/
(exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-10.5)*(rx-10.5))),;

L_d4 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+9.5)*(rx+9.5)) +
exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+13.5)*(rx+13.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-9.5)*(rx-9.5)) +
exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-13.5)*(rx-13.5)))/
(exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+8.5)*(rx+8.5)) +
exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+11.5)*(rx+11.5)) +
exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+15.5)*(rx+15.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-7.5)*(rx-7.5))+exp(n*(rx-8.5)*(rx-8.5)) +
exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-11.5)*(rx-11.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-15.5)*(rx-15.5))),;

```

COMPUTER PROGRAM LISTING APPENDIX

```

D1_data[i]      = L_d0;
D1_data[i+1]    = L_d1;
D1_parity[i]    = L_d2;
D2_parity[i]    = L_d3;
D1_parity[i+1]  = L_d4;

/* symbol 1, I dimension: 16AM */
d0 = data_d[i+2];
d1 = Enc1[i+2];
d2 = Enc2[i+2];
d3 = Enc2[i+1];
tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
tx = (d0 == 0 ? (tx - 4) : (4 - tx));
rx = tx + SIGMA_16AM_of_39_512QAM * gasdev();
n = (-1.0) / (2 * SIGMA_16AM_of_39_512QAM * SIGMA_16AM_of_39_512QAM);
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
            (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
            (exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
            exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
            exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
            (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
            exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
            exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
            (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
            exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i+2]    = L_d0;
D1_parity[i+2]  = L_d1;
D2_parity[i+2]  = L_d2;
D2_parity[i+1]  = L_d3;
i = i+2;
}
/*
 * interleave data:
 */
r_ileav(D1_data, rule);

#endif

/*mio*/
#ifndef R12_16QAM
/*
 * Channel: we transmit two 4-AM symbols to emulate a 16-QAM symbol.

```

COMPUTER PROGRAM LISTING APPENDIX

```

* 2 info bits and 2 parity bits are mapped to 1 16-QAM symbols which in
* turn are simulated as 2 4-AM symbols to achieve 2bit/s/Hz
* d0 is MSB and d1 is LSB in a 4-AM:(d0,d1) = 01--00-|-10--11
*                                         -3   -1    1    3
* INT_SIZE to be a multiple of 2
*/
n = (-1.0) / (2 * SIGMA_12_16QAM * SIGMA_12_16QAM);
for(i = 0; i < INT_SIZE; i++)
{
    /* symbol 1 */
    d0 = data[i];
    d1 = Enc1[i];
    tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
    rx = tx + SIGMA_12_16QAM * gasdev();
    L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /
                (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3)))); 
    L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3))) /
                (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1)))); 
    D1_data[i] = L_d0;
    D1_parity[i] = L_d1;
    D2_parity[i] = 0.0;

    /* symbol 2 */
    d0 = data[i+1];
    d1 = Enc2[i+1];
    tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
    rx = tx + SIGMA_12_16QAM * gasdev();
    L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /
                (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3)))); 
    L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3))) /
                (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1)))); 
    D1_data[i+1] = L_d0;
    D2_parity[i+1] = L_d1;
    D1_parity[i+1] = 0.0;

    i = i+1;
}
#endif

#ifndef R34_16QAM
/*
 * Channel: we transmit two 4-AM symbols to emulate a 16-QAM symbol.
 * 6 info bits and 2 parity bits are mapped to 2 16-QAM symbols which in
 * turn are simulated as 4 4-AM symbols to achieve 3bit/s/Hz
 * d0 is MSB and d1 is LSB in a 4-AM:(d0,d1) = 01--00-|-10--11
 *                                         -3   -1    1    3
 * INT_SIZE to be a multiple of 6
*/
n = (-1.0) / (2 * SIGMA_34_16QAM * SIGMA_34_16QAM);
for(i = 0; i < INT_SIZE; i++)
{
    /* symbol 1 */
    d0 = data[i];
    d1 = data[i+1];
    tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
    rx = tx + SIGMA_34_16QAM * gasdev();
    L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /
                (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3)))); 
    L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3))) /
                (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1)))); 
    D1_data[i] = L_d0;
    D1_data[i+1] = L_d1;

    /* symbol 2 */
    d0 = data[i+2];
    d1 = Enc1[i+1];
    tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
    rx = tx + SIGMA_34_16QAM * gasdev();
    L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /

```

COMPUTER PROGRAM LISTING APPENDIX

```

        (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3))));  

L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3)))/  

           (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1))));  

D1_data[i+2] = L_d0;  

D1_parity[i+1] = L_d1;  

  

/* symbol 3 */  

d0 = data[i+3];  

d1 = data[i+4];  

tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;  

rx = tx + SIGMA_34_16QAM * gasdev();  

L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3)))/  

           (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3))));  

L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3)))/  

           (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1))));  

D1_data[i+3] = L_d0;  

D1_data[i+4] = L_d1;  

  

/* symbol 4 */  

d0 = data[i+5];  

d1 = Enc2[i+4];  

tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;  

rx = tx + SIGMA_34_16QAM * gasdev();  

L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3)))/  

           (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3))));  

L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3)))/  

           (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1))));  

D1_data[i+5] = L_d0;  

D2_parity[i+4] = L_d1;  

D1_parity[i] = 0.0;  

D1_parity[i+2] = 0.0;  

D1_parity[i+3] = 0.0;  

D1_parity[i+4] = 0.0;  

D1_parity[i+5] = 0.0;  

D2_parity[i] = 0.0;  

D2_parity[i+1] = 0.0;  

D2_parity[i+2] = 0.0;  

D2_parity[i+3] = 0.0;  

D2_parity[i+4] = 0.0;  

D2_parity[i+5] = 0.0;  

i = i+5;  

    }  

#endif  

  

#ifndef R56_64QAM  

/*
 * Channel:  

 * d0 is MSB and d2 is LSB in 8AM:(d0,d1,d2):  

 *      010---011---001---000---100---101---111---110  

 *      -3.5   -2.5   -1.5   -0.5   0.5   1.5   2.5   3.5
 */
/*  

 * Channel: we transmit two 8AM symbols to emulate a 64QAM symbol.  

 * 10 info bits and 2 parity bits are mapped to 2 64QAM symbols which in  

 * turn are simulated as 4 8AM symbols to achieve 5bit/s/Hz.
 */
/*  

 * INT_SIZE to be a multiple of 10
*/
n = (-1.0) / (2 * SIGMA_56_64QAM * SIGMA_56_64QAM);
for(i = 0; i < INT_SIZE; i++)
{
    /* symbol 1 */
    d0 = data[i];
    d1 = data[i+1];
    d2 = Enc1[i];
    tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
    rx = tx + SIGMA_56_64QAM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
               exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
               exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));  

}

```

COMPUTER PROGRAM LISTING APPENDIX

```

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
            (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
            (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

D1_data[i] = L_d0;
D1_data[i+1] = L_d1;
D1_parity[i] = L_d2;
D1_parity[i+1] = 0;
D2_parity[i] = 0;
D2_parity[i+1] = 0;

/* symbol 2 */
d0 = data[i+2];
d1 = data[i+3];
d2 = data[i+4];
tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
rx = tx + SIGMA_56_64QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
            (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
            (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
            (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

D1_data[i+2] = L_d0;
D1_data[i+3] = L_d1;
D1_data[i+4] = L_d2;
D1_parity[i+2] = 0;
D1_parity[i+3] = 0;
D1_parity[i+4] = 0;
D2_parity[i+2] = 0;
D2_parity[i+3] = 0;
D2_parity[i+4] = 0;

/* symbol 3 */
d0 = data[i+5];
d1 = data[i+6];
d2 = Enc2[i+5];
tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
rx = tx + SIGMA_56_64QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
            (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
            (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
            (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

D1_data[i+5] = L_d0;
D1_data[i+6] = L_d1;
D2_parity[i+5] = L_d2;
D2_parity[i+6] = 0;
D1_parity[i+5] = 0;

```

COMPUTER PROGRAM LISTING APPENDIX

```

D1_parity[i+6] = 0;

/* symbol 4 */
d0 = data[i+7];
d1 = data[i+8];
d2 = data[i+9];
tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
rx = tx + SIGMA_56_64QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
            (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
            (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
            (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

D1_data[i+7] = L_d0;
D1_data[i+8] = L_d1;
D1_data[i+9] = L_d2;
D1_parity[i+7] = 0;
D1_parity[i+8] = 0;
D1_parity[i+9] = 0;
D2_parity[i+7] = 0;
D2_parity[i+8] = 0;
D2_parity[i+9] = 0;

i = i+9;
}
#endif

#endif /* R57_64QAM */

#endif /* R57_128QAM */

/* Q dimension:
 * d0 is MSB and d2 is LSB in 8-AM: (d0,d1,d2):
 *      010---011---001---000---100---101---111---110
 *      -3.5   -2.5   -1.5   -0.5    0.5    1.5    2.5    3.5
 * I dimension:
 * d0 is MSB and d3 is LSB in 16AM: (d0,d1,d2,d3):
 *
 *      0010---0011---0001---0000---0100---0101---0111---0110
 *      -7.5    -6.5    -5.5    -4.5    -3.5    -2.5    -1.5    -0.5
 *
 *      1010---1011---1001---1000---1100---1101---1111---1110
 *      7.5     6.5     5.5     4.5     3.5     2.5     1.5     0.5
 *
 * INT_SIZE to be a multiple of 5
 */

for(i = 0; i < INT_SIZE; i++)
{
    /* Q dimension: 8AM */
    d0 = data[i];
    d1 = data[i+1];
    d2 = Enc1[i];
    tx = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
    rx = tx + SIGMA_8AM_of_128QAM * gasdev();
    n = (-1.0) / (2 * SIGMA_8AM_of_128QAM * SIGMA_8AM_of_128QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

    L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
                (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

    D1_data[i+7] = L_d0;
    D1_data[i+8] = L_d1;
    D1_data[i+9] = L_d2;
    D1_parity[i+7] = 0;
    D1_parity[i+8] = 0;
    D1_parity[i+9] = 0;
    D2_parity[i+7] = 0;
    D2_parity[i+8] = 0;
    D2_parity[i+9] = 0;
}
```

COMPUTER PROGRAM LISTING APPENDIX

```

(exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
 exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
 exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
 (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
 exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

D1_data[i] = L_d0;
D1_data[i+1] = L_d1;
D1_parity[i] = L_d2;
D1_parity[i+1] = 0.0;
D1_parity[i+2] = 0.0;
D1_parity[i+3] = 0.0;
D1_parity[i+4] = 0.0;

/* I dimension: 16AM */
d0 = data[i+2];
d1 = data[i+3];
d2 = data[i+4];
d3 = Enc2[i];
tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
tx = (d0 == 0 ? (tx - 4) : (4 - tx));
rx = tx + SIGMA_16AM_of_128QAM * gasdev();
n = (-1.0) / (2 * SIGMA_16AM_of_128QAM * SIGMA_16AM_of_128QAM);
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
 exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
 exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
 exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
 (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
 exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
 exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
 exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
 exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
 exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
 exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
 (exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
 exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
 exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
 exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
 exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
 exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
 exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
 (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
 exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
 exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
 exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
 exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
 exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
 exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
 (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
 exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
 exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
 exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i+2] = L_d0;
D1_data[i+3] = L_d1;
D1_data[i+4] = L_d2;
D2_parity[i] = L_d3;
D2_parity[i+1] = 0.0;
D2_parity[i+2] = 0.0;
D2_parity[i+3] = 0.0;
D2_parity[i+4] = 0.0;

i = i+4;
}

```

COMPUTER PROGRAM LISTING APPENDIX

```

#endif

#ifndef R58_256QAM
/*
 * Channel:
 * d0 is MSB and d3 is LSB in 16AM:(d0,d1,d2,d3):
 *
 *      0010---0011---0001---0000---0100---0101---0111---0110
 *      -7.5   -6.5   -5.5   -4.5   -3.5   -2.5   -1.5   -0.5
 *
 *      1010---1011---1001---1000---1100---1101---1111---1110
 *      7.5    6.5    5.5    4.5    3.5    2.5    1.5    0.5
 *
 * Channel: we transmit two 16AM symbols to emulate a 256QAM symbol.
 * 10 info bits and 6 parity bits are mapped to 2 256QAM symbols which in
 * turn are simulated as 4 16AM symbols to achieve 6bit/s/Hz.
 *
 * INT_SIZE to be a multiple of 10
 */
n = (-1.0) / (2 * SIGMA_58_256QAM * SIGMA_58_256QAM);

/*
 * deinterleave data:
 */
for(i = 0; i < INT_SIZE; i++)
    data_d[i] = data[i];
r_deileava(data_d, rule);

for(i = 0; i < INT_SIZE; i++)
{
    /* symbol 1 */
    d0 = data_d[i];
    d1 = data_d[i+1];
    d2 = data_d[i+2];
    d3 = Encl[i];
    tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
    tx = (d0 == 0 ? (tx - 4): (4 - tx));
    /* Test the mapping to the 16AM constellation:
     * if (i < 500)
     *     printf("\n(%d%d%d%d) = %f", (int)d0, (int)d1, (int)d2, (int)d3, tx);
     */
    rx = tx + SIGMA_58_256QAM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
                exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));
    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
                exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));
    L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
               (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));
```

COMPUTER PROGRAM LISTING APPENDIX

```

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
            exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
            (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
            exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i]      = L_d0;
D1_data[i+1]    = L_d1;
D1_data[i+2]    = L_d2;
D1_parity[i]    = L_d3;
D1_parity[i+1]  = 0;
D1_parity[i+2]  = 0;
D2_parity[i]    = 0;
D2_parity[i+1]  = 0;

/* symbol 2 */
d0 = data_d[i+3];
d1 = data_d[i+4];
d2 = Enc2[i+2];
d3 = Enc1[i+4];
tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
tx = (d0 == 0 ? (tx - 4) : (4 - tx));
rx = tx + SIGMA_58_256QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
            (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
            (exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
            exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
            exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
            (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
            exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
            exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
            (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
            exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i+3]    = L_d0;
D1_data[i+4]    = L_d1;
D2_parity[i+2]  = L_d2;
D1_parity[i+4]  = L_d3;
D1_parity[i+3]  = 0;
D2_parity[i+3]  = 0;
D2_parity[i+4]  = 0;

/* symbol 3 */
d0 = data_d[i+5];

```

COMPUTER PROGRAM LISTING APPENDIX

```

d1 = data_d[i+6];
d2 = data_d[i+7];
d3 = Enc2[i+5];
tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
tx = (d0 == 0 ? (tx - 4): (4 - tx));
rx = tx + SIGMA_58_256QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
(exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
(exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i+5] = L_d0;
D1_data[i+6] = L_d1;
D1_data[i+7] = L_d2;
D2_parity[i+5] = L_d3;
D2_parity[i+6] = 0;
D2_parity[i+7] = 0;
D1_parity[i+5] = 0;
D1_parity[i+6] = 0;

/* symbol 4 */
d0 = data_d[i+8];
d1 = data_d[i+9];
d2 = Enc1[i+7];
d3 = Enc2[i+9];
tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
tx = (d0 == 0 ? (tx - 4): (4 - tx));
rx = tx + SIGMA_58_256QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/

```

COMPUTER PROGRAM LISTING APPENDIX

```

(exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
 exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
 exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
 exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
 exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
 exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
 exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
 (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
 exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
 exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
 exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
 exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
 exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
 exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
 (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
 exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
 exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
 exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i+8] = L_d0;
D1_data[i+9] = L_d1;
D1_parity[i+7] = L_d2;
D2_parity[i+9] = L_d3;
D1_parity[i+8] = 0;
D1_parity[i+9] = 0;
D2_parity[i+7] = 0;
D2_parity[i+8] = 0;

    i = i+9;
}
/*
 * interleave data:
 */
r_ileav(D1_data, rule);
#endif

#endif R68_256QAM
/*
 * Channel:
 * d0 is MSB and d3 is LSB in 16AM:(d0,d1,d2,d3):
 *
 *      0010---0011---0001---0000---0100---0101---0111---0110
 *      -7.5   -6.5   -5.5   -4.5   -3.5   -2.5   -1.5   -0.5
 *
 *      1010---1011---1001---1000---1100---1101---1111---1110
 *      7.5    6.5    5.5    4.5    3.5    2.5    1.5    0.5
 *
 * Channel: we transmit two 16AM symbols to emulate a 256QAM symbol.
 * 6 info bits and 2 parity bits are mapped to one 256QAM symbol which in
 * turn is simulated as 2 16AM symbols to achieve 6bit/s/Hz.
 *
 * INT_SIZE to be a multiple of 6
 */
n = (-1.0) / (2 * SIGMA_68_256QAM * SIGMA_68_256QAM);

/*
 * deinterleave data:
 */
for(i = 0; i < INT_SIZE; i++)
    data_d[i] = data[i];
r_deileava(data_d, rule);

for(i = 0; i < INT_SIZE; i++)
{
    /* symbol 1 */
    d0 = data_d[i];
    d1 = data_d[i+1];
}

```

COMPUTER PROGRAM LISTING APPENDIX

```

d2 = data_d[i+2];
d3 = Enc1[i];
tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
tx = (d0 == 0 ? (tx - 4): (4 - tx));
rx = tx + SIGMA_68_256QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
(exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
(exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i] = L_d0;
D1_data[i+1] = L_d1;
D1_data[i+2] = L_d2;
D1_parity[i] = L_d3;
D1_parity[i+1] = 0;
D1_parity[i+2] = 0;
D2_parity[i] = 0;
D2_parity[i+1] = 0;
D2_parity[i+2] = 0;

/* symbol 2 */
d0 = data_d[i+3];
d1 = data_d[i+4];
d2 = data_d[i+5];
d3 = Enc2[i+3];
tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
tx = (d0 == 0 ? (tx - 4): (4 - tx));
rx = tx + SIGMA_68_256QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));
```

COMPUTER PROGRAM LISTING APPENDIX

```

exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
(exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
 exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
 exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
 exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
 exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
 exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
 exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
 exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
 exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
 exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
 exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
 exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
 exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
(exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
 exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
 exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
 exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i+3] = L_d0;
D1_data[i+4] = L_d1;
D1_data[i+5] = L_d2;
D2_parity[i+3] = L_d3;
D2_parity[i+4] = 0;
D2_parity[i+5] = 0;
D1_parity[i+3] = 0;
D1_parity[i+4] = 0;
D1_parity[i+5] = 0;

i = i+5;
}

/*
 * interleave data:
 */
r_ileav(D1_data, rule);

#endif

#ifndef R69_512QAM
/*
 * Interleaver should be a multiple of 12, e.g., 6144
 *
 * Q dimension:
 * d0 is MSB and d4 is LSB in 32AM: (d0,d1,d2,d3,d4):
 * 00010--00011--00001--00000--00100--00101--00111--00110
 * -15.5 -14.5 -13.5 -12.5 -11.5 -10.5 -9.5 -8.5
 *
 * 01010--01011--01001--01000--01100--01101--01111--01110
 * -0.5 -1.5 -2.5 -3.5 -4.5 -5.5 -6.5 -7.5
 *
 * 11010--11011--11001--11000--11100--11101--11111--11110
 * 0.5 1.5 2.5 3.5 4.5 5.5 6.5 7.5
 *
 * 10010--10011--10001--10000--10100--10101--10111--10110
 * 15.5 14.5 13.5 12.5 11.5 10.5 9.5 8.5
 *
 * I dimension:
 * d0 is MSB and d3 is LSB in 16AM: (d0,d1,d2,d3):
 *
 * 0010---0011---0001---0000---0100---0101---0111---0110
 * -7.5 -6.5 -5.5 -4.5 -3.5 -2.5 -1.5 -0.5
 *
 * 1010---1011---1001---1000---1100---1101---1111---1110
 * 7.5 6.5 5.5 4.5 3.5 2.5 1.5 0.5

```

COMPUTER PROGRAM LISTING APPENDIX

```

/*
 * deinterleave data:
 */
for(i = 0; i < INT_SIZE; i++)
    data_d[i] = data[i];
r_deileava(data_d, rule);

for(i = 0; i < INT_SIZE; i++)
{
    /* symbol 1, Q dimension: 32AM */
    d0 = data_d[i];
    d1 = data_d[i+1];
    d2 = data_d[i+2];
    d3 = data_d[i+3];
    d4 = Enc1[i];
    tx = 2*d2 - 2*d3 + 4*d2*d3 - 1.0 + (((2*d2-1)*(2*d3-1))<0?(d4-0.5):(0.5-d4));
    tx = (d1 == 0 ? (tx - 4) : (4 - tx));
    tx = (d0 == 0 ? (tx - 8) : (8 - tx));
    rx = tx + SIGMA_32AM_of_512QAM * gasdev();
    n = (-1.0) / (2 * SIGMA_32AM_of_512QAM * SIGMA_32AM_of_512QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
        exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
        exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
        exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
        exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
        exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)) +
        exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
        exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)))/
        (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
        exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
        exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
        exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
        exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
        exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
        exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
        exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5))));

    L_d1 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
        exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
        exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
        exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
        exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
        exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
        exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
        exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)))/
        (exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
        exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)) +
        exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
        exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)) +
        exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
        exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
        exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
        exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5))));

    L_d2 = log((exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
        exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
        exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
        exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
        exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
        exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
        exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
        exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)))/
        (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
        exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
        exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
        exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
        exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
        exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
        exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
        exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)));
}

```

COMPUTER PROGRAM LISTING APPENDIX

```

exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5))));

L_d3 = log((exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)))/
(exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-10.5)*(rx-10.5))));

L_d4 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+9.5)*(rx+9.5)) +
exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+13.5)*(rx+13.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-9.5)*(rx-9.5)) +
exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-13.5)*(rx-13.5)))/
(exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+8.5)*(rx+8.5)) +
exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+11.5)*(rx+11.5)) +
exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+15.5)*(rx+15.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-7.5)*(rx-7.5))+exp(n*(rx-8.5)*(rx-8.5)) +
exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-11.5)*(rx-11.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-15.5)*(rx-15.5))));

D1_data[i]      = L_d0;
D1_data[i+1]    = L_d1;
D1_data[i+2]    = L_d2;
D1_data[i+3]    = L_d3;
D1_parity[i]    = L_d4;
D1_parity[i+1]  = 0.0;
D1_parity[i+2]  = 0.0;
D1_parity[i+3]  = 0.0;
D1_parity[i+5]  = 0.0;

/* symbol 1, I dimension: 16AM */
d0 = data_d[i+4];
d1 = data_d[i+5];
d2 = Enc1[i+4];
d3 = Enc2[i+2];
tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
tx = (d0 == 0 ? (tx - 4) : (4 - tx));
rx = tx + SIGMA_16AM_of_512QAM * gasdev();
n = (-1.0) / (2 * SIGMA_16AM_of_512QAM * SIGMA_16AM_of_512QAM);
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
(exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-10.5)*(rx-10.5))));
```

COMPUTER PROGRAM LISTING APPENDIX

```

exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
(exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i+4] = L_d0;
D1_data[i+5] = L_d1;
D1_parity[i+4] = L_d2;
D2_parity[i+2] = L_d3;
D2_parity[i] = 0.0;
D2_parity[i+1] = 0.0;
D2_parity[i+3] = 0.0;
D2_parity[i+4] = 0.0;
D2_parity[i+5] = 0.0;

/* symbol 2, Q dimension: 32AM */
d0 = data_d[i+6];
d1 = data_d[i+7];
d2 = data_d[i+8];
d3 = data_d[i+9];
d4 = Enc2[i+6];
tx = 2*d2 - 2*d3 + 4*d2*d3 - 1.0 + (((2*d2-1)*(2*d3-1))<0?(d4-0.5):(0.5-d4));
tx = (d1 == 0 ? (tx - 4) : (4 - tx));
tx = (d0 == 0 ? (tx - 8) : (8 - tx));
rx = tx + SIGMA_32AM_of_512QAM * gasdev();
n = (-1.0) / (2 * SIGMA_32AM_of_512QAM * SIGMA_32AM_of_512QAM);
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)) +
exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5))));

L_d1 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)))/
(exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)) +
exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)) +
exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5))));
```

COMPUTER PROGRAM LISTING APPENDIX

```

exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)));;

L_d2 = log((exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5))));;

L_d3 = log((exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)))/
(exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-10.5)*(rx-10.5))));;

L_d4 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+9.5)*(rx+9.5)) +
exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+13.5)*(rx+13.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-9.5)*(rx-9.5)) +
exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-13.5)*(rx-13.5)))/
(exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+8.5)*(rx+8.5)) +
exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+11.5)*(rx+11.5)) +
exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+15.5)*(rx+15.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-7.5)*(rx-7.5))+exp(n*(rx-8.5)*(rx-8.5)) +
exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-11.5)*(rx-11.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-15.5)*(rx-15.5))));;

D1_data[i+6] = L_d0;
D1_data[i+7] = L_d1;
D1_data[i+8] = L_d2;
D1_data[i+9] = L_d3;
D2_parity[i+6] = L_d4;
D2_parity[i+7] = 0.0;
D2_parity[i+8] = 0.0;
D2_parity[i+9] = 0.0;
D2_parity[i+11] = 0.0;

/* symbol 2, I dimension: 16AM */
d0 = data_d[i+10];
d1 = data_d[i+11];
d2 = Enc2[i+10];

```

COMPUTER PROGRAM LISTING APPENDIX

```

d3 = Encl[i+8];
tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
tx = (d0 == 0 ? (tx - 4) : (4 - tx));
rx = tx + SIGMA_16AM_of_512QAM * gasdev();
n = (-1.0) / (2 * SIGMA_16AM_of_512QAM * SIGMA_16AM_of_512QAM);
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
(exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-7.5)*(rx-7.5)))/
(exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
(exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i+10] = L_d0;
D1_data[i+11] = L_d1;
D2_parity[i+10] = L_d2;
D1_parity[i+8] = L_d3;
D1_parity[i+6] = 0.0;
D1_parity[i+7] = 0.0;
D1_parity[i+9] = 0.0;
D1_parity[i+10] = 0.0;
D1_parity[i+11] = 0.0;

i = i+11;
}
/*
 * interleave data:
 */
r_ileav(D1_data, rule);

#endif

#ifndef R710_1024QAM
/*
 * Use S2044_33_1 interleaver (multiple of 14)
 *
 * I and Q dimensions:
 * d0 is MSB and d4 is LSB in 32AM:(d0,d1,d2,d3,d4):
 *      00010--00011--00001--00000--00100--00101--00111--00110
 *      -15.5   -14.5   -13.5   -12.5   -11.5   -10.5   -9.5   -8.5
 *
 *      01010--01011--01001--01000--01100--01101--01111--01110

```

COMPUTER PROGRAM LISTING APPENDIX

```

*      -0.5   -1.5   -2.5   -3.5   -4.5   -5.5   -6.5   -7.5
*
*      11010--11011--11001--11000--11100--11101--11111--11110
*      0.5    1.5    2.5    3.5    4.5    5.5    6.5    7.5
*
*      10010--10011--10001--10000--10100--10101--10111--10110
*      15.5   14.5   13.5   12.5   11.5   10.5   9.5    8.5
*
*/
n    = (-1.0) / (2 * SIGMA_710_1024QAM * SIGMA_710_1024QAM);
for(i = 0; i < INT_SIZE; i++)
{
    /* symbol 1, Q dimension */
    d0 = data[i];
    d1 = data[i+1];
    d2 = data[i+2];
    d3 = data[i+3];
    d4 = Enc1[i];
    tx = 2*d2 - 2*d3 + 4*d2*d3 - 1.0 + (((2*d2-1)*(2*d3-1))<0?(d4-0.5):(0.5-d4));
    tx = (d1 == 0 ? (tx - 4) : (4 - tx));
    tx = (d0 == 0 ? (tx - 8) : (8 - tx));
    rx = tx + SIGMA_710_1024QAM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
                exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
                exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)) +
                exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
                exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)))/
                (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
                exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
                exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
                exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
                exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
                exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)));
    L_d1 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
                exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
                exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))/
                (exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
                exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)) +
                exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
                exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)) +
                exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
                exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
                exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
                exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)));
    L_d2 = log((exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
                exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
                exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
                exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
                exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)))/
                (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
                exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
                exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +

```

COMPUTER PROGRAM LISTING APPENDIX

COMPUTER PROGRAM LISTING APPENDIX

```

exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5))),;

L_d1 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)))/
(exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)) +
exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)) +
exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5))),;

L_d2 = log((exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5))),;

L_d3 = log((exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)))/
(exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-10.5)*(rx-10.5))),;

L_d4 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+9.5)*(rx+9.5)) +
exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+13.5)*(rx+13.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-9.5)*(rx-9.5)) +
exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-13.5)*(rx-13.5)))/
(exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+8.5)*(rx+8.5)) +
exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+11.5)*(rx+11.5)) +
exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+15.5)*(rx+15.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-7.5)*(rx-7.5))+exp(n*(rx-8.5)*(rx-8.5)) +
exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-11.5)*(rx-11.5)) +
exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5))),;

```

COMPUTER PROGRAM LISTING APPENDIX

COMPUTER PROGRAM LISTING APPENDIX

COMPUTER PROGRAM LISTING APPENDIX

```

exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)));;

L_d1 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)))/
(exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)) +
exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)) +
exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5));;

L_d2 = log((exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)))/
(exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5));;

L_d3 = log((exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)))/
(exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-10.5)*(rx-10.5));;

L_d4 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+5.5)*(rx+5.5)) +
exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+9.5)*(rx+9.5)) +
exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+13.5)*(rx+13.5)) +
exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)) +
exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-9.5)*(rx-9.5)) +
exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-13.5)*(rx-13.5)))/
(exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+3.5)*(rx+3.5)) +
exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+8.5)*(rx+8.5)) +
exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+11.5)*(rx+11.5)) +
exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+15.5)*(rx+15.5)) +
exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-3.5)*(rx-3.5)) +
exp(n*(rx-7.5)*(rx-7.5))+exp(n*(rx-8.5)*(rx-8.5)) +

```

COMPUTER PROGRAM LISTING APPENDIX

```

exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-11.5)*(rx-11.5)) +
exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-15.5)*(rx-15.5)));
D1_data[i+11] = L_d0;
D1_data[i+12] = L_d1;
D1_data[i+13] = L_d2;
D2_parity[i+10] = L_d3;
D1_parity[i+10] = L_d4;
D1_parity[i+7] = 0.0;
D1_parity[i+8] = 0.0;
D1_parity[i+9] = 0.0;
D1_parity[i+11] = 0.0;
D1_parity[i+12] = 0.0;
D1_parity[i+13] = 0.0;

i = i+13;
}
#endif

/***********************/

/*
 * At this moment we received the whole turbo code block:
 * D1_data[] stores the received information sequence,
 * D1_parity[] stores the received punctured parity P sequence and
 * D2_data[] stores the interleaved received information sequence,
 */
for(i = 0; i < INT_SIZE; i++)
    D2_data[i] = D1_data[i];
r_ileav(D2_data, rule);
/*
 * D2_parity[] stores the received punctured parity Q sequence.
 */
for(iteration = 1; iteration <= NR_ITER; iteration++)
{
    /*
     * Start one iteration of the turbo decoder here:
     */
#ifndef R46_64QAM_TTCM_VoCAL
    jat_map2(jat_code1, D1_data, D1_parity, D1_app, D1_exi);
#else
    jat_map1(jat_code1, D1_data, D1_parity, D1_app, D1_exi);
#endif
    /*
     * Interleave the extrinsic information from Decoder1:
     */
    for(k = 0; k < INT_SIZE; k++)
        D2_app[k] = D1_exi[k];
    r_ileav(D2_app, rule);

    /*
     * Decoder2:
     */
#ifndef R46_64QAM_TTCM_VoCAL
    jat_map2(jat_code2, D2_data, D2_parity, D2_app, D2_exi);
#else
    jat_map1(jat_code2, D2_data, D2_parity, D2_app, D2_exi);
#endif
    /*
     * Deinterleave the extrinsic information from Decoder2:
     */
    r_deileav(D2_exi, rule);
    for(k = 0; k < INT_SIZE; k++)
        D1_app[k] = D2_exi[k];
}

```

COMPUTER PROGRAM LISTING APPENDIX

```

#ifndef R46_64QAM_TTCM_VoCAL
    for(k = 0; k < INT_SIZE/2; k++)
        Dec_data[k] = D1_data[k] + log(D1_exi[k]) + log(D2_exi[k]);

    /*
     * Re-encode with encoder1:
     */
    jat_code1->enc_state = 0;           /* reset encoder1's state   */
    for(k = 0; k < INT_SIZE/2; k++)
        Enc1[k] = jat_enc_bp_fp(jat_code1, ((Dec_data[k] > 0.0)?1:0));

    /*
     * interleave data:
     */
    for(k = 0; k < INT_SIZE/2; k++)
        data_i[k] = ((Dec_data[k] > 0.0)?1:0);
    r_ileava(data_i, rule);

    /*
     * Re-encode with encoder2:
     */
    jat_code2->enc_state = 0;           /* reset encoder2's state   */
    for(k = 0; k < INT_SIZE/2)
        Enc2[k] = jat_enc_bp_fp(jat_code2, data_i[k]);

    /*
     * Find the closest point out of four in the sub-constellation
     */
    for(k = 0; k < INT_SIZE/2 - 1; )
    {
        u4 = ((Dec_data[k] > 0.0)?1:0);
        u3 = ((Dec_data[k+1] > 0.0)?1:0);
        u2 = Enc1[k];
        u1 = Enc2[k+1];

        rx_I = D1_data[k + INT_SIZE/2];
        rx_Q = D1_data[k + INT_SIZE/2 + 1];

        j = 4*u4+8*u3+16*u2+32*u1;
        v00_I = find_tx_I(j);
        v00_Q = find_tx_Q(j);
        v01_I = find_tx_I(j+1);
        v01_Q = find_tx_Q(j+1);
        v10_I = find_tx_I(j+2);
        v10_Q = find_tx_Q(j+2);
        v11_I = find_tx_I(j+3);
        v11_Q = find_tx_Q(j+3);

        Dec_data[k+INT_SIZE/2] = log(
            (exp(n*((rx_I-v11_I)*(rx_I-v11_I)+(rx_Q-v11_Q)*(rx_Q-v11_Q)))+
             exp(n*((rx_I-v10_I)*(rx_I-v10_I)+(rx_Q-v10_Q)*(rx_Q-v10_Q))))/
            (exp(n*((rx_I-v01_I)*(rx_I-v01_I)+(rx_Q-v01_Q)*(rx_Q-v01_Q)))+
             exp(n*((rx_I-v00_I)*(rx_I-v00_I)+(rx_Q-v00_Q)*(rx_Q-v00_Q)))));
        Dec_data[k+INT_SIZE/2+1] = log(
            (exp(n*((rx_I-v11_I)*(rx_I-v11_I)+(rx_Q-v11_Q)*(rx_Q-v11_Q)))+
             exp(n*((rx_I-v01_I)*(rx_I-v01_I)+(rx_Q-v01_Q)*(rx_Q-v01_Q))))/
            (exp(n*((rx_I-v10_I)*(rx_I-v10_I)+(rx_Q-v10_Q)*(rx_Q-v10_Q)))+
             exp(n*((rx_I-v00_I)*(rx_I-v00_I)+(rx_Q-v00_Q)*(rx_Q-v00_Q)))));
        k = k+2;
    }

#else
    for(k = 0; k < INT_SIZE; k++)
        Dec_data[k] = D1_data[k] + log(D1_exi[k]) + log(D2_exi[k]);
#endif

/*

```

COMPUTER PROGRAM LISTING APPENDIX

```

        * print errors:
        */
k = print_err(data, Dec_data, iteration, block, no_err);
if(k == 0)
    break;

/*
 * End one iteration of the turbo decoder here.
 */
}

}

free(jat_code1->P0state);
free(jat_code1->P1state);
free(jat_code1->N0state);
free(jat_code1->N1state);
free(jat_code1->Coded0);
free(jat_code1->Coded1);
free(jat_code1);
free(jat_code2->P0state);
free(jat_code2->P1state);
free(jat_code2->N0state);
free(jat_code2->N1state);
free(jat_code2->Coded0);
free(jat_code2->Coded1);
free(jat_code2);
free(rule);
free(data);
free(data_i);
free(data_d);
free(no_err);
free(Enc1);
free(Enc2);
free(D1_data);
free(D1_parity);
free(D1_app);
free(D1_exi);
free(D2_data);
free(D2_parity);
free(D2_app);
free(D2_exi);
free(Dec_data);
free(frame_hist);
free(Zero_data);
for(i = THRESHOLD_ITER; i <= NR_ITER; i++)
    free(bit_hist_array[i]);
free(bit_hist_array);
free(bit_hist_block);
}

/*****************/
/* jat_trellis_bp_fp() initializes the code structure           */
void jat_trellis_bp_fp(jat_code *code_str)
{
    int i;
    for(i = 0; i < code_str->nr_states ; i++)
    {
        code_str->enc_state  = i;
        code_str->P0state[i] = jat_ps(code_str, 0);
        code_str->P1state[i] = jat_ps(code_str, 1);
        code_str->enc_state  = i;
        code_str->Coded0[i]  = jat_enc_bp_fp(code_str, 0);
        code_str->N0state[i] = code_str->enc_state; /*next state i if d = 0 */
        code_str->enc_state  = i;
        code_str->Coded1[i]  = jat_enc_bp_fp(code_str, 1);
        code_str->N1state[i] = code_str->enc_state; /*next state i if d = 1 */
    }
}

```

COMPUTER PROGRAM LISTING APPENDIX

```

/*****************************************/
/* jat_ps() returns the previous state, given the code structure & the input */
/*      bit for the previous state                                         */
/*      input:   current state, input bit for the previous state          */
/*      output:  previous state                                         */
int jat_ps(jat_code *code_st, int inp)
{
    int pr_state, pr_msb, i, j, k, l;

    if(code_st->enc_mem == 1)
    {
        pr_state = code_st->enc_state ^ inp;
    }
    else
    {
        /*find previous state:                                              */
        pr_msb = (inp & 0x1) ^ (code_st->enc_state & 0x1);
        for(i=1,j=2,k=(1      <<      code_st->enc_mem-1),l=code_st->enc_mem-1;i<code_st-
>enc_mem;i++,l--)
        {
            pr_msb = pr_msb ^ (((code_st->enc_state&j)>>i) & ((code_st->bp&k)>>l));
            j = j << 1;
            k = k >> 1;
        }

        pr_state = ((code_st->enc_state >> 1) & ((l<<(code_st->enc_mem - 1)) - 1)) |
        (pr_msb << (code_st->enc_mem - 1));
    }
    return (pr_state);
}

/*****************************************/
/* jat_enc_bp_fp() - rate 1/2 systematic feedback convolutional enc.      */
/*      input:  input bit to be encoded                                     */
/*      output: the coded bit                                             */
/* Note:   the lsb of the enc_state will have the new input bit           */
/*      the msb of the enc_state matches the lsb of bp & fp               */
int jat_enc_bp_fp(jat_code *code_st, int data)
{
    int new_lsb, parity, i, j, k, l;

    new_lsb = data;
    if(code_st->enc_mem == 1)
    {
        parity = code_st->enc_state ^ data;
        code_st->enc_state = parity;
    }
    else
    {
        /* xor it with the bits of the enc_state1 for which bp1 is one      */
        for(i = 0, j = 1, k = (1 << code_st->enc_mem-1), l = code_st->enc_mem - 1; i <
code_st->enc_mem; i++, l--)
        {
            new_lsb = new_lsb ^ (((code_st->enc_state&j)>>i) & ((code_st->bp&k)>>l));
            j = j << 1;
            k = k >> 1;
        }

        /* find the parity bit                                                 */
        parity = new_lsb & ((code_st->fp&(l<<code_st->enc_mem)) >> code_st->enc_mem);
        for(i = 0, j = 1, k = (1 << code_st->enc_mem-1), l = code_st->enc_mem - 1; i <
code_st->enc_mem; i++, l--)
        {
            parity = parity ^ (((code_st->enc_state&j)>>i) & ((code_st->fp&k)>>l));
            j = j << 1;
            k = k >> 1;
        }
    }
}

```

COMPUTER PROGRAM LISTING APPENDIX

```

        /* update code_st->enc_state
           code_st->enc_state = ((code_st->enc_state & (code_st->nr_states/2 - 1)) << 1) |
new_lsb;
}
return (parity);
}

/*********************************************
void r_ileav(double *array, int *rule_i)
{
    double *i_wmem;
    int k;

    i_wmem = (double *)malloc(sizeof(double) * INT_SIZE);
    if(i_wmem == 0)
        printf("\nCouldn't allocate i_wmem memory!");
    for(k = 0; k < INT_SIZE; k++)
        i_wmem[k] = array[k];
    for(k = 0; k < INT_SIZE; k++)
        array[k] = i_wmem[rule_i[2*k+1]];
    free(i_wmem);
}

/*********************************************
void r_ileava(int *array, int *rule_i)
{
    int *i_wmem;
    int k;

    i_wmem = (int *)malloc(sizeof(int) * INT_SIZE);
    if(i_wmem == 0)
        printf("\nCouldn't allocate i_wmem memory!");
    for(k = 0; k < INT_SIZE; k++)
        i_wmem[k] = array[k];
    for(k = 0; k < INT_SIZE; k++)
        array[k] = i_wmem[rule_i[2*k+1]];
    free(i_wmem);
}

/*********************************************
void r_deileav(double *array, int *rule_d)
{
    double *d_wmem;
    int k;

    d_wmem = (double *)malloc(sizeof(double) * INT_SIZE);
    if(d_wmem == 0)
        printf("\nCouldn't allocate d_wmem memory!");
    for(k = 0; k < INT_SIZE; k++)
        d_wmem[rule_d[2*k+1]] = array[k];
    for(k = 0; k < INT_SIZE; k++)
        array[k] = d_wmem[k];
    free(d_wmem);
}

/*********************************************
void r_deileava(int *array, int *rule_d)
{
    int *d_wmem;
    int k;

    d_wmem = (int *)malloc(sizeof(int) * INT_SIZE);
    if(d_wmem == 0)
        printf("\nCouldn't allocate d_wmem memory!");
    for(k = 0; k < INT_SIZE; k++)
        d_wmem[rule_d[2*k+1]] = array[k];
    for(k = 0; k < INT_SIZE; k++)

```

COMPUTER PROGRAM LISTING APPENDIX

```

        array[k] = d_wmem[k];
        free(d_wmem);
    }

/************************************************************************/
/* nrgen() returns a random number between 0 and 1                  */
/*          (uniform distribution generator)                         */
/************************************************************************/
double nrgen()
{
    long z, k;

    k = s1 / 53668;
    s1 = 40014 * (s1 - k * 53668) - k * 12211;
    if(s1 < 0)
        s1 += 2147483563;
    k = s2 / 52774;
    s2 = 40692 * (s2 - k * 52774) - k * 3791;
    if(s2 < 0)
        s2 += 2147483399;
    z = s1 - s2;
    if(z < 1)
        z += 2147483562;
    return ((double)z / 2147483563);
}

/************************************************************************/
/* nrgenbin() returns a 0 or 1 (uniform distribution)                 */
/************************************************************************/
int nrgenbin()
{
    return ((nrgen() > 0.5)?1:0);
}

/************************************************************************/
/* gasdev() returns a normally distributed deviate                   */
/*          with zero mean and unit variance                         */
/************************************************************************/
double gasdev()
{
    static int      iset = 0;
    static double   gset;
    double         fac, r, v1, v2;

    if(iset == 0)
    {
        /* pick two uniform numbers in the square extending from      */
        /* -1 to +1 in each direction, see if they are in the          */
        /* unit circle, and if they are not, try again.                  */
        do
        {
            v1 = 2.0 * nrgen() - 1.0;
            v2 = 2.0 * nrgen() - 1.0;
            r = v1 * v1 + v2 * v2;
        }
        while (r >= 1.0 || r == 0.0);
        fac = sqrt(-2.0 * log(r)/r);
        /* now make the Box-Muller transformation to get two normal   */
        /* deviates; return one and save the other for next time.       */
        gset = v1 * fac;
        iset = 1;           /* set flag
        return (v2 * fac);
    }
    else
    {
        /* we have an extra deviate handy, so unset the flag and     */
        /* return it.
        iset = 0;
        return (gset);
    }
}

```

COMPUTER PROGRAM LISTING APPENDIX

```

*****
/* errors() returns the nr. of positions in which two blocks of data are      */
/* different; it accepts a shift between the addresses                      */
/* inputs: the address of the first block of integers                      */
/*          the address of the second block of doubles                     */
/*          the size of the block (blocks are equal)                         */
/* output: the number of positions in which the two blocks are dif. */ */
int errors(int *block1, double *block2, int size, int iter_nr)
{
    int i;
    int count = 0;

    for(i = 0; i < size; i++)
        if(block1[i] != ((block2[i] > 0.0)?1:0))
    {
        count++;
#if defined BIT_HIST
        if(iter_nr>=THRESHOLD_ITER)
        {
            *bit_hist_array[iter_nr] = bit_hist_block[iter_nr];
            bit_hist_array[iter_nr]++;
            *bit_hist_array[iter_nr] = i;
            bit_hist_array[iter_nr]++;
        }
#endif
    }
#if defined BIT_HIST
    if((count>0)&&(iter_nr >= THRESHOLD_ITER))
        bit_hist_block[iter_nr]++;
#endif
    return (count);
}

*****
/* print_err() append to the file the nr. of errors and BER                  */
/* returns: number of bit errors in a block                                     */
int print_err(int *data1, double *data2, int iter_no, int block_no, int *err)
{
    int i, j, nr;
    int block_err = 0;
    char fname[] = BIT_HIST_FILE_NAME;
    char sss[] = {'0','1','2','3','4','5','6','7','8','9'};
    FILE *out_file = NULL;
    int *pi;

    if((iter_no == 1) && (block_no == 1))
    {
        out_file = fopen(ERROR_FILE_NAME, "a");
        if(!out_file)
        {
            printf("Error2: the output file could not be opened!\n");
            exit (1);
        }
        fprintf(out_file, "ref_tc.c, RSC1_enc_mem = %d, RSC1_fp = %d, RSC1_bp = %d,
RSC2_enc_mem = %d, RSC2_fp = %d, RSC2_bp = %d, s1 = %d, s2 = %d, int_size = %d, Limit soft
outputs to = %e, Eb/No = %fdb\n", RSC1_ENC_MEM, RSC1_FP, RSC1_BP, RSC2_ENC_MEM, RSC2_FP,
RSC2_BP, SEED1, SEED2, INT_SIZE, (double) MAX, (double) EBNO);
        fclose(out_file);
    }
    block_err = errors(data1, data2, INT_SIZE, iter_no);
    err[iter_no - 1] += block_err;

    ++frame_hist[(iter_no-1)*(INT_SIZE+1) + block_err];

    if((iter_no == NR_ITER) && (block_err != 0))
        ++frame_err;
}

```

COMPUTER PROGRAM LISTING APPENDIX

```

#if defined BIT_HIST
if((block_err != 0) && (iter_no >= THRESHOLD_ITER))
{
    fname[strlen(fname)-1] = sss[iter_no];
    out_file = fopen(fname, "a");
    if(!out_file)
    {
        printf("Error: the bit_hist file could not be opened!\n");
        exit (1);
    }
    for(pi = bit_hist_array[NR_ITER +iter_no]; pi < bit_hist_array[iter_no]; pi = pi+2)
        fprintf(out_file, "\n%06d %06d", *pi, *(pi+1));
    fclose(out_file);
    bit_hist_array[iter_no] = bit_hist_array[NR_ITER +iter_no];
}
#endif

if(((iter_no == NR_ITER) && (((block_no % PRINT_BLOCKS) == 0) || (err[NR_ITER - 1] >
MAX_ERRORS))) || (((block_no % PRINT_BLOCKS) == 0) && (block_err == 0)))
{

    out_file = fopen(ERROR_FILE_NAME, "a");
    if(!out_file)
    {
        printf("Error3: the output file could not be opened!\n");
        exit (1);
    }
    nr = block_no * INT_SIZE;
    fprintf(out_file, "\n\nNr. of info bits: %d (%d blocks)",
            nr, block_no);
    for(j = 0; j < NR_ITER; j++)
        fprintf(out_file, "\niter: %02d, Errors: %06d, BER = %e",
                j + 1, err[j], (double)err[j]/INT_SIZE/block_no);
    total_err = err[NR_ITER - 1];
    fprintf(out_file, "\nFrame error = %f(%f errors per block)\n",
            (double)frame_err/block_no,
            (frame_err == 0?0.0:(double)err[NR_ITER-1]/frame_err));
    fclose(out_file);

    if((block_no % 100000) == 0)
    {
        out_file = fopen(FRAME_HIST_FILE_NAME, "a");
        if(!out_file)
        {
            printf("Error4: the .fhist file could not be opened!\n");
            exit (1);
        }
        nr = block_no * INT_SIZE;
        fprintf(out_file, "\n\nNr. of info bits: %d (%d blocks)",
                nr, block_no);
        for(j = 0; j <= INT_SIZE; j++)
            fprintf(out_file, "\n%03d %03d",
                    j, frame_hist[(NR_ITER-1)*(INT_SIZE+1)+j]);
        fclose(out_file);
    }
}
return (block_err);
}

/*****************************************/
/* This is a MAP decoder for a cs->nr_states states jat_code. */
/* function: decodes a block of received data of length INT_SIZE. */
/*           It assumes that the encoder state starts from state zero */
/* input:    code structure, I address, Q address, L_in address */
/* output:   the extrinsic information in L_out */
/* globals:  noise */
*/

```

COMPUTER PROGRAM LISTING APPENDIX

```

/*
*****
* As jat_map but outputs probability and not log(probability)
* It also can handle very large interleavers
*/
void jat_map1(jat_code *cs, double *I, double *Q, double *L_in, double *L_out)
{
    double    sum, sum_0, sum_1, max;
    int      i, j, k, st;
    double   *alpha_old;
    double   *alpha_new;
    double   *beta0;
    double   *beta1;
    double   *probI;
    double   *probQ;

alpha_old = (double *)malloc(sizeof(double) * 2 * cs->nr_states);
alpha_new = (double *)malloc(sizeof(double) * 2 * cs->nr_states);

beta0 = (double *)malloc(sizeof(double) * INT_SIZE * cs->nr_states);
if(beta0 == 0)
{
    printf("Couldn't allocate beta0 memory!\n");
    exit(1);
}

beta1 = (double *)malloc(sizeof(double) * INT_SIZE * cs->nr_states);
if(beta1 == 0)
{
    printf("Couldn't allocate beta1 memory!\n");
    exit(1);
}

probI = (double *)malloc(sizeof(double) * INT_SIZE);
if(probI == 0)
{
    printf("Couldn't allocate probI memory!\n");
    exit(1);
}

probQ = (double *)malloc(sizeof(double) * INT_SIZE);
if(probQ == 0)
{
    printf("Couldn't allocate probQ memory!\n");
    exit(1);
}

/* initialize the alpha_old metrics
for(st = 0; st < cs->nr_states; st++)                                */
    for(k = 0; k < 2; k++)
        *(alpha_old + k * cs->nr_states + st) = 0.0;

*(alpha_old + cs->P0state[0]) = 1.0;
*(alpha_old + cs->nr_states + cs->P1state[0]) = 1.0;

/* initialize beta's
for(st = 0; st < cs->nr_states; st++)                                */
{
    beta0[(INT_SIZE - 1) * cs->nr_states + st] = 1.0;
    beta1[(INT_SIZE - 1) * cs->nr_states + st] = 1.0;
}

/* compute all beta's
for(i = INT_SIZE - 2; i >= 0; i--)                                     */
{
    probI[i + 1] = exp(I[i + 1]) * L_in[i + 1];
    probQ[i + 1] = exp(Q[i + 1]);
    for(st = 0; st < cs->nr_states; st++)
    {
        /* compute beta0[i][st]:                                         */

```

COMPUTER PROGRAM LISTING APPENDIX

```

        beta0[i * cs->nr_states + st] = beta0[(i + 1) * cs->nr_states + cs-
>N0state[st]]*
            ((cs->Coded0[cs->N0state[st]] == 0)?1:probQ[i + 1]) +
            beta1[(i + 1) * cs->nr_states + cs->N0state[st]]*probI[i + 1]*
            ((cs->Coded1[cs->N0state[st]] == 0)?1:probQ[i + 1]);
        betal[i * cs->nr_states + st] = beta0[(i + 1) * cs->nr_states + cs-
>N1state[st]]*
            ((cs->Coded0[cs->N1state[st]] == 0)?1:probQ[i + 1]) +
            beta1[(i + 1) * cs->nr_states + cs->N1state[st]]*probI[i + 1]*
            ((cs->Coded1[cs->N1state[st]] == 0)?1:probQ[i + 1]);
    }

    max = beta0[i * cs->nr_states];
    for(st = 1; st < cs->nr_states; st++)
        if(beta0[i * cs->nr_states + st] > max)
            max = beta0[i * cs->nr_states + st];
    for(st = 0; st < cs->nr_states; st++)
        if(betal[i * cs->nr_states + st] > max)
            max = betal[i * cs->nr_states + st];
    for(st = 0; st < cs->nr_states; st++)
    {
        beta0[i * cs->nr_states + st] = beta0[i * cs->nr_states + st] / max;
        betal[i * cs->nr_states + st] = betal[i * cs->nr_states + st] / max;
    }
}

/* now we have all beta's; we can compute alpha for all states for each      */
/* data bit and using beta's we compute lambda                                */
probI[0] = exp(I[0]) * L_in[0];
probQ[0] = exp(Q[0]);
for(k = 0; k < INT_SIZE; k++)
{
    for(st = 0; st < cs->nr_states; st++)
    {
        sum = *(alpha_old + cs->P0state[st]) + *(alpha_old + cs->nr_states + cs-
>P1state[st]);
        *(alpha_new + st) = sum * ((cs->Coded0[st] == 0)?1:probQ[k]);
        *(alpha_new + cs->nr_states + st) = sum * probI[k] * ((cs->Coded1[st] ==
0)?1:probQ[k]);
    }

    /* find the max value and renormalize alpha's:                            */
    max = *alpha_new;
    for(st = 0; st < cs->nr_states; st++)
        for(j = 0; j < 2; j++)
            if(*(alpha_new + cs->nr_states * j + st) > max)
                max = *(alpha_new + cs->nr_states * j + st);
    for(st = 0; st < cs->nr_states; st++)
        for(j = 0; j < 2; j++)
            *(alpha_new + cs->nr_states * j + st) = *(alpha_new + cs->nr_states * j + st)/
max;

    /* find sum_0 and sum_1 over all states for L_out:                      */
    sum_0 = 0.0;
    sum_1 = 0.0;
    for(st = 0; st < cs->nr_states; st++)
    {
        sum_0 += *(alpha_new + st) * beta0[k * cs->nr_states + st];
        sum_1 += *(alpha_new + cs->nr_states + st) * betal[k * cs->nr_states + st];
    }

    /* output the extrinsic information:                                       */
    L_out[k] = (sum_1 / sum_0) / exp(I[k]) / L_in[k];
    if(L_out[k] > MAX)
        L_out[k] = MAX;
    if(L_out[k] < 1/MAX)
        L_out[k] = 1/MAX;

    for(st = 0; st < cs->nr_states; st++)
        for(j = 0; j < 2; j++)/* update alphas                         */
            *(alpha_old + cs->nr_states * j + st)=*(alpha_new + cs->nr_states * j + st);
}

```

COMPUTER PROGRAM LISTING APPENDIX

```
        }
        free(beta0);
        free(beta1);
        free(probI);
        free(probQ);
        free(alpha_old);
        free(alpha_new);
    }

double find_tx_I(int k)
{
    double tx_I;
    switch(k)
    {
        case 0:
            tx_I = 0.5;
            break;
        case 1:
            tx_I = -3.5;
            break;
        case 2:
            tx_I = 0.5;
            break;
        case 3:
            tx_I = -3.5;
            break;
        case 4:
            tx_I = 2.5;
            break;
        case 5:
            tx_I = -1.5;
            break;
        case 6:
            tx_I = 2.5;
            break;
        case 7:
            tx_I = -1.5;
            break;
        case 8:
            tx_I = 2.5;
            break;
        case 9:
            tx_I = -1.5;
            break;
        case 10:
            tx_I = 2.5;
            break;
        case 11:
            tx_I = -1.5;
            break;
        case 12:
            tx_I = 0.5;
            break;
        case 13:
            tx_I = -3.5;
            break;
        case 14:
            tx_I = 0.5;
            break;
        case 15:
            tx_I = -3.5;
            break;
        case 16:
            tx_I = 1.5;
            break;
        case 17:
            tx_I = -2.5;
            break;
    }
}
```

COMPUTER PROGRAM LISTING APPENDIX

```
case 18:
    tx_I = 1.5;
    break;
case 19:
    tx_I = -2.5;
    break;
case 20:
    tx_I = 3.5;
    break;
case 21:
    tx_I = -0.5;
    break;
case 22:
    tx_I = 3.5;
    break;
case 23:
    tx_I = -0.5;
    break;
case 24:
    tx_I = 3.5;
    break;
case 25:
    tx_I = -0.5;
    break;
case 26:
    tx_I = 3.5;
    break;
case 27:
    tx_I = -0.5;
    break;
case 28:
    tx_I = 1.5;
    break;
case 29:
    tx_I = -2.5;
    break;
case 30:
    tx_I = 1.5;
    break;
case 31:
    tx_I = -2.5;
    break;
case 32:
    tx_I = 1.5;
    break;
case 33:
    tx_I = -2.5;
    break;
case 34:
    tx_I = 1.5;
    break;
case 35:
    tx_I = -2.5;
    break;
case 36:
    tx_I = 3.5;
    break;
case 37:
    tx_I = -0.5;
    break;
case 38:
    tx_I = 3.5;
    break;
case 39:
    tx_I = -0.5;
    break;
case 40:
    tx_I = 3.5;
    break;
case 41:
    tx_I = -0.5;
```

COMPUTER PROGRAM LISTING APPENDIX

```
        break;
case 42:
    tx_I = 3.5;
    break;
case 43:
    tx_I = -0.5;
    break;
case 44:
    tx_I = 1.5;
    break;
case 45:
    tx_I = -2.5;
    break;
case 46:
    tx_I = 1.5;
    break;
case 47:
    tx_I = -2.5;
    break;
case 48:
    tx_I = 0.5;
    break;
case 49:
    tx_I = -3.5;
    break;
case 50:
    tx_I = 0.5;
    break;
case 51:
    tx_I = -3.5;
    break;
case 52:
    tx_I = 2.5;
    break;
case 53:
    tx_I = -1.5;
    break;
case 54:
    tx_I = 2.5;
    break;
case 55:
    tx_I = -1.5;
    break;
case 56:
    tx_I = 2.5;
    break;
case 57:
    tx_I = -1.5;
    break;
case 58:
    tx_I = 2.5;
    break;
case 59:
    tx_I = -1.5;
    break;
case 60:
    tx_I = 0.5;
    break;
case 61:
    tx_I = -3.5;
    break;
case 62:
    tx_I = 0.5;
    break;
case 63:
    tx_I = -3.5;
    break;
}
return(tx_I);
}
```

COMPUTER PROGRAM LISTING APPENDIX

```
double find_tx_Q(int k)
{
    double tx_Q;
    switch(k)
    {
        case 0:
            tx_Q = 2.5;
            break;
        case 1:
            tx_Q = 2.5;
            break;
        case 2:
            tx_Q = -1.5;
            break;
        case 3:
            tx_Q = -1.5;
            break;
        case 4:
            tx_Q = 0.5;
            break;
        case 5:
            tx_Q = 0.5;
            break;
        case 6:
            tx_Q = -3.5;
            break;
        case 7:
            tx_Q = -3.5;
            break;
        case 8:
            tx_Q = 2.5;
            break;
        case 9:
            tx_Q = 2.5;
            break;
        case 10:
            tx_Q = -1.5;
            break;
        case 11:
            tx_Q = -1.5;
            break;
        case 12:
            tx_Q = 0.5;
            break;
        case 13:
            tx_Q = 0.5;
            break;
        case 14:
            tx_Q = -3.5;
            break;
        case 15:
            tx_Q = -3.5;
            break;
        case 16:
            tx_Q = 3.5;
            break;
        case 17:
            tx_Q = 3.5;
            break;
        case 18:
            tx_Q = -0.5;
            break;
        case 19:
            tx_Q = -0.5;
            break;
        case 20:
            tx_Q = 1.5;
            break;
        case 21:
            tx_Q = 1.5;
```

COMPUTER PROGRAM LISTING APPENDIX

```
        break;
case 22:
    tx_Q = -2.5;
    break;
case 23:
    tx_Q = -2.5;
    break;
case 24:
    tx_Q = 3.5;
    break;
case 25:
    tx_Q = 3.5;
    break;
case 26:
    tx_Q = -0.5;
    break;
case 27:
    tx_Q = -0.5;
    break;
case 28:
    tx_Q = 1.5;
    break;
case 29:
    tx_Q = 1.5;
    break;
case 30:
    tx_Q = -2.5;
    break;
case 31:
    tx_Q = -2.5;
    break;
case 32:
    tx_Q = 2.5;
    break;
case 33:
    tx_Q = 2.5;
    break;
case 34:
    tx_Q = -1.5;
    break;
case 35:
    tx_Q = -1.5;
    break;
case 36:
    tx_Q = 0.5;
    break;
case 37:
    tx_Q = 0.5;
    break;
case 38:
    tx_Q = -3.5;
    break;
case 39:
    tx_Q = -3.5;
    break;
case 40:
    tx_Q = 2.5;
    break;
case 41:
    tx_Q = 2.5;
    break;
case 42:
    tx_Q = -1.5;
    break;
case 43:
    tx_Q = -1.5;
    break;
case 44:
    tx_Q = 0.5;
    break;
case 45:
```

COMPUTER PROGRAM LISTING APPENDIX

```

        tx_Q = 0.5;
        break;
case 46:
    tx_Q = -3.5;
    break;
case 47:
    tx_Q = -3.5;
    break;
case 48:
    tx_Q = 3.5;
    break;
case 49:
    tx_Q = 3.5;
    break;
case 50:
    tx_Q = -0.5;
    break;
case 51:
    tx_Q = -0.5;
    break;
case 52:
    tx_Q = 1.5;
    break;
case 53:
    tx_Q = 1.5;
    break;
case 54:
    tx_Q = -2.5;
    break;
case 55:
    tx_Q = -2.5;
    break;
case 56:
    tx_Q = 3.5;
    break;
case 57:
    tx_Q = 3.5;
    break;
case 58:
    tx_Q = -0.5;
    break;
case 59:
    tx_Q = -0.5;
    break;
case 60:
    tx_Q = 1.5;
    break;
case 61:
    tx_Q = 1.5;
    break;
case 62:
    tx_Q = -2.5;
    break;
case 63:
    tx_Q = -2.5;
    break;
}
return(tx_Q);
}

/*************************************************/
/* This is a MAP decoder for a cs->nr_states states jat_code.          */
/*   function: decodes a block of received data of length INT_SIZE/2.      */
/*   input:      It assumes that the encoder state starts from state zero  */
/*   output:     code structure, I address, Q address, L_in address         */
/*   globals:    noise                                                       */
/*                                                       */

/*

```

COMPUTER PROGRAM LISTING APPENDIX

```
*****
* As jat_map but outputs probability and not log(probability)
* It also can handle very large interleavers
*/
void jat_map2(jat_code *cs, double *I, double *Q, double *L_in, double *L_out)
{
    double    sum, sum_0, sum_1, max;
    int       i, j, k, st;
    double   *alpha_old;
    double   *alpha_new;
    double   *beta0;
    double   *beta1;
    double   *probI;
    double   *probQ;

    alpha_old = (double *)malloc(sizeof(double) * 2 * cs->nr_states);
    alpha_new = (double *)malloc(sizeof(double) * 2 * cs->nr_states);

    beta0 = (double *)malloc(sizeof(double) * INT_SIZE/2 * cs->nr_states);
    if(beta0 == 0)
    {
        printf("Couldn't allocate beta0 memory!\n");
        exit(1);
    }

    beta1 = (double *)malloc(sizeof(double) * INT_SIZE/2 * cs->nr_states);
    if(beta1 == 0)
    {
        printf("Couldn't allocate beta1 memory!\n");
        exit(1);
    }

    probI = (double *)malloc(sizeof(double) * INT_SIZE/2);
    if(probI == 0)
    {
        printf("Couldn't allocate probI memory!\n");
        exit(1);
    }

    probQ = (double *)malloc(sizeof(double) * INT_SIZE/2);
    if(probQ == 0)
    {
        printf("Couldn't allocate probQ memory!\n");
        exit(1);
    }

    /* initialize the alpha_old metrics
    for(st = 0; st < cs->nr_states; st++)
        for(k = 0; k < 2; k++)
            *(alpha_old + k * cs->nr_states + st) = 0.0;

    *(alpha_old + cs->P0state[0]) = 1.0;
    *(alpha_old + cs->nr_states + cs->P1state[0]) = 1.0;

    /* initialize beta's
    for(st = 0; st < cs->nr_states; st++)
    {
        beta0[(INT_SIZE/2 - 1) * cs->nr_states + st] = 1.0;
        beta1[(INT_SIZE/2 - 1) * cs->nr_states + st] = 1.0;
    }

    /* compute all beta's
    for(i = INT_SIZE/2 - 2; i >= 0; i--)
    {
        probI[i + 1] = exp(I[i + 1]) * L_in[i + 1];
        probQ[i + 1] = exp(Q[i + 1]);
        for(st = 0; st < cs->nr_states; st++)
        {
            /* compute beta0[i][st]: */
            */
            /*
            beta0[i][st] = probI[i + 1] * beta0[(INT_SIZE/2 - 1) * cs->nr_states + st] +
                           probQ[i + 1] * beta1[(INT_SIZE/2 - 1) * cs->nr_states + st];
            beta1[i][st] = probI[i + 1] * beta1[(INT_SIZE/2 - 1) * cs->nr_states + st] +
                           probQ[i + 1] * beta0[(INT_SIZE/2 - 1) * cs->nr_states + st];
        }
    }
}
```

COMPUTER PROGRAM LISTING APPENDIX

```

        beta0[i * cs->nr_states + st] = beta0[(i + 1) * cs->nr_states + cs-
>N0state[st]]*
            ((cs->Coded0[cs->N0state[st]] == 0)?1:probQ[i + 1])+
            betal[(i + 1) * cs->nr_states + cs->N0state[st]]*probI[i + 1]*
            ((cs->Coded1[cs->N0state[st]] == 0)?1:probQ[i + 1]);
        betal[i * cs->nr_states + st] = beta0[(i + 1) * cs->nr_states + cs-
>N1state[st]]*
            ((cs->Coded0[cs->N1state[st]] == 0)?1:probQ[i + 1])+
            betal[(i + 1) * cs->nr_states + cs->N1state[st]]*probI[i + 1]*
            ((cs->Coded1[cs->N1state[st]] == 0)?1:probQ[i + 1]);
    }

    max = beta0[i * cs->nr_states];
    for(st = 1; st < cs->nr_states; st++)
        if(beta0[i * cs->nr_states + st] > max)
            max = beta0[i * cs->nr_states + st];
    for(st = 0; st < cs->nr_states; st++)
        if(betal[i * cs->nr_states + st] > max)
            max = betal[i * cs->nr_states + st];
    for(st = 0; st < cs->nr_states; st++)
    {
        beta0[i * cs->nr_states + st] = beta0[i * cs->nr_states + st] / max;
        betal[i * cs->nr_states + st] = betal[i * cs->nr_states + st] / max;
    }
}

/* now we have all beta's; we can compute alpha for all states for each */
/* data bit and using beta's we compute lambda */
probI[0] = exp(I[0]) * L_in[0];
probQ[0] = exp(Q[0]);
for(k = 0; k < INT_SIZE/2; k++)
{
    for(st = 0; st < cs->nr_states; st++)
    {
        sum = *(alpha_old + cs->P0state[st]) + *(alpha_old + cs->nr_states + cs-
>P1state[st]);
        *(alpha_new + st) = sum * ((cs->Coded0[st] == 0)?1:probQ[k]);
        *(alpha_new + cs->nr_states + st) = sum * probI[k] * ((cs->Coded1[st] ==
0)?1:probQ[k]);
    }
}

/* find the max value and renormalize alpha's: */
max = *alpha_new;
for(st = 0; st < cs->nr_states; st++)
    for(j = 0; j < 2; j++)
        if(*(alpha_new + cs->nr_states * j + st) > max)
            max = *(alpha_new + cs->nr_states * j + st);
for(st = 0; st < cs->nr_states; st++)
    for(j = 0; j < 2; j++)
        *(alpha_new + cs->nr_states * j + st) = *(alpha_new + cs->nr_states * j + st)/
max;

/* find sum_0 and sum_1 over all states for L_out: */
sum_0 = 0.0;
sum_1 = 0.0;
for(st = 0; st < cs->nr_states; st++)
{
    sum_0 += *(alpha_new + st) * beta0[k * cs->nr_states + st];
    sum_1 += *(alpha_new + cs->nr_states + st) * betal[k * cs->nr_states + st];
}

/* output the extrinsic information: */
L_out[k] = (sum_1 / sum_0) / exp(I[k]) / L_in[k];
if(L_out[k] > MAX)
    L_out[k] = MAX;
if(L_out[k] < 1/MAX)
    L_out[k] = 1/MAX;

for(st = 0; st < cs->nr_states; st++)
    for(j = 0; j < 2; j++)/* update alphas */
        *(alpha_old + cs->nr_states * j + st)=*(alpha_new + cs->nr_states * j + st);

```

COMPUTER PROGRAM LISTING APPENDIX

```
    }
    free(beta0);
    free(beta1);
    free(probI);
    free(probQ);
    free(alpha_old);
    free(alpha_new);
}
```

COMPUTER PROGRAM LISTING APPENDIX

interlever.c

```
#define MAX_CINDEX 46
#define MAX_RINDEX 47
#define MAX_ELEMENT 2100
#include <stdio.h>
#include <stdlib.h>

void main (void)
{
    int ra, ca; //Ia sequence row and column indices
    int count; //Counter for each bit in DMT frame
    int element; //Element number used for finding if element within array
    FILE *output;

    output=fopen("interleaver","w");
    //Initial sequence indices

    ra=MAX_RINDEX-1;
    ca=0;

    //Adjust the initial indices for Ia if beyond ending element
    element=ra*MAX_CINDEX+ca;

    while (element >=MAX_ELEMENT) {
        ra--;
        ca++;
        if (ra<0) {
            ra=MAX_RINDEX-1;
            ca=ca+(MAX_RINDEX-1);
        }
        ca=ca%MAX_CINDEX;

        element= ra*MAX_CINDEX+ra;
    }

    //Fetch all elements in sequence Ia
    for (count = 0; count<MAX_ELEMENT; count++) {
        //Fetch array[ra][ca]
        element=ra*MAX_CINDEX+ca;
        fprintf(output,"%d %d\n",count,element);
        //Update indices for next access

        do {
            ra--;
            ca++;
            if (ra<0) {
                ra=MAX_RINDEX-1;
                ca=ca+(MAX_RINDEX-1);
            }
            ca=ca%MAX_CINDEX;

            element = ra * MAX_CINDEX+ca;
        } while (element >= MAX_ELEMENT);
    }
}
```

COMPUTER PROGRAM LISTING APPENDIX

S-type interleaver generator

```

program int(input,output);
{This program generates mod-k S-random and symmetric mod-k S-random interleavers.

const Nmax = 65536; {maximum interleaver size}

var G,H,I,J,K,L,M,N,S,count,temp,prt,i_,j_,k_,im,jm:longint;
inta,hat,deinti:array[0..Nmax] of longint;
pass,good:boolean;
s1,s2:longint; {seeds for function uniform}
into,deinto:text;
sym:char;

function max(x,y:longint):longint;
{Finds the maximum of x and y}
begin{max}
  if x > y
    then max := x
    else max := y;
end;{max}

function min(x,y:longint):longint;
{Finds the minimum of x and y}
begin{min}
  if x < y
    then min := x
    else min := y;
end;{min}

function uniform(var s1,s2:longint):double;
{Generates a random number from 0.0 < x < 1.0}
const m0 = 2147483562;
      m1 = 2147483563;
      m2 = 2147483399;
      a1 = 40014;
      a2 = 40692;
      q1 = 53668;
      q2 = 52774;
      r1 = 12211;
      r2 = 3791;
var k:longint;
begin(uniform)
  k := s1 div q1;
  s1 := a1*(s1-k*q1) - k*r1;
  if s1 < 0 then s1 := s1+m1;

  k := s2 div q2;
  s2 := a2*(s2-k*q2) - k*r2;
  if s2 < 0 then s2 := s2+m2;

  k := s1-s2;
  if k < 1 then k := k+m0;
  uniform := k/m1;
end;{uniform}

procedure srandom;
{Generates mod-k S-random interleaver}
label 98;

procedure reject;
{reject random number}
begin(reject)
  count := count-1;
  if count = 0
    then begin(bad int)
          good := false;
          goto 98;
        end;{bad int}

```

COMPUTER PROGRAM LISTING APPENDIX

```

pass := false;
for M := K to count-1 do
    hat[M] := hat[M+1];
    hat[count] := J;
end;{reject}

begin{S-random}
repeat
    writeln('seed1 = ',s1:1,', seed2 = ',s2:1);
    good := true;
    for I := 0 to N-1 do
        hat[I] := I;
    for I := 0 to N-1 do
        begin{make int}
            count := N-I;
            i_ := I mod k_;
            im := min(i_,k_-i_);
        repeat
            pass := true;
            K := trunc(count*uniform(s1,s2));
            if K = count then K := K-1;
            J := hat[K];
            if k_ > 1 then
                begin{mod k test}
                    j_ := J mod k_;
                    jm := min(j_,k_-j_);
                    if im <> jm then reject;
                end;{mod k test}
            if pass = true then
                begin{S-random test}
                    for L := max(0,I-S) to I-1 do
                        if (abs(J-inta[L]) <= S) and (pass = true) then reject;
                end{S-random test}
        until pass = true;
        for M := K to N-I-2 do
            hat[M] := hat[M+1];
            inta[I] := J;
        end;{make int}
    98:
    until good = true;
end{S-random};

procedure trandom;
{Generates symmetric mod-k S-random interleaver}
label 99;

procedure rejects;
{reject random number}
begin{reject S}
    count := count-1;
    if count = 0 then
        begin{bad int}
            good := false;
            goto 99;
        end;{bad int}
    pass := false;
    inta[I] := -1;
    inta[J] := -1;
    for M := K to count-1 do
        hat[M] := hat[M+1];
        hat[count] := J;
    end;{reject S}

procedure test;
{S-random test}
begin{test}
    if (inta[L] >= 0) and (abs(G-inta[L]) <= S) then rejects;
    L := L+1;
end;{test}

begin{T-random}

```

COMPUTER PROGRAM LISTING APPENDIX

```

repeat
  writeln('seed1 = ',s1:1,', seed2 = ',s2:1);
  good := true;
  for I := 0 to N-1 do
    begin{init}
      hat[I] := I;
      inta[I] := -1;
    end;{init}
  H := N;
  I := 0;
  repeat
    count := H;
    while (inta[I] >= 0) and (I < N) do I := I+1;
    i_ := I mod k_;
    im := min(i_,k_-i_);
    repeat
      pass := true;
      K := trunc(count*uniform(s1,s2));
      if K = count then K := K-1;
      J := hat[K];
      if k_ > 1 then
        begin{mod k test}
          j_ := J mod k_;
          jm := min(j_,k_-j_);
          if im <> jm then rejects;
        end;{mod k test}
      if pass = true then
        begin{S-random test}
          inta[I] := J;
          inta[J] := I;
          G := J;
          L := max(0,I-S);
          while (pass = true) and (L < I) do test;
          L := I+1;
          while (pass = true) and (L < min(I+S,N)) do test;
          G := I;
          L := max(0,J-S);
          while (pass = true) and (L < J) do test;
          L := J+1;
          while (pass = true) and (L < min(J+S,N)) do test;
        end;{S-random test}
      until pass = true;

      H := H-1;
      for M := K to H-1 do
        hat[M] := hat[M+1];
      if I <> J then
        begin{sym}
          K := 0;
          while (hat[K] <> I) and (K < H) do K := K+1;
          H := H-1;
          for M := K to H-1 do
            hat[M] := hat[M+1];
        end;{sym}
      until H = 0;
    99:
    until good = true;
  end{T-random};

begin{int}
  s1 := 12345; {initialise seeds for uniform}
  s2 := 67890;

  writeln;
  writeln('Random Interleaver Generator V1.01');
  writeln('Copyright (c) 1998 Small World Communications. All rights reserved.');
  writeln;
  write('Enter block size N <= ',Nmmax:1,' : ');
  readln(N);
  write('Enter S parameter (S=1 is random) : ');
  readln(S);

```

COMPUTER PROGRAM LISTING APPENDIX

```

write('Enter mod-k parameter (k=1 is normal): ');
readln(k_);
repeat
  write('Do you want a symmetric interleaver? ');
  readln(sym);
  pass := (sym = 'y') or (sym = 'Y') or (sym = 'n') or (sym = 'N');
  if pass = false then
    writeln('Invalid entry. Try again.');
until pass = true;
writeln;
case sym of
  'y','Y': trandom;
  'n','N': srandom;
end;{case}

assign(into,'int.dat');
rewrite(into);
for I := 0 to N-1 do
  writeln(into,inta[I]:1);
close(into);

for I := 0 to N-1 do
  hat[I] := 0;
for I := 0 to N-1 do
  begin{test}
    J := inta[I];
    hat[J] := hat[J] + 1;
  end;{test}
pass := true;
for I := 0 to N-1 do
  if hat[I] < 1 then pass := false;
if pass = false
  then writeln('Bad interleaver!');

pass := true;
I := 0;
repeat
  J := inta[I];
  for L := max(0,I-S) to I-1 do
    if (abs(J-inta[L]) <= S) and (pass = true)
      then begin
        pass := false;
        writeln('Interleaver failed S-test');
        writeln(I:1,' ',inta[I]:1,' ',L:1,' ',inta[L]:1);
      end;
  I := I+1;
until (pass = false) or (I = N);

K := 0;
for I := 0 to N-1 do
  K := max(K,abs(I-inta[I]));
writeln('Dmin = ',K:1);
writeln('Interleaver table int.dat successfully generated');
end.{int}

```